User Manual

3-Heights™ PDF Security API

Version 4.5

PDF/A compliant

PDF-TOOLS.COM
Premium PDF Technology

Contents

1 Introduction . . . 1
1.1 Description . . . 1
1.2 Functions . . . 2
1.3 Interfaces . . . 3
1.4 Operating Systems . . . 3
1.5 How to Best Read this Manual . . . 3
1.6 Digital Signatures . . . 3
What is an Electronic Signature? . . . 4
How to Create Electronic Signatures . . . 6
2 Installation and Deployment . . . 7
2.1 Windows . . . 7
2.2 Unix . . . 7
2.3 Interfaces . . . 9
2.4 Interface Specific Installation Steps . . . 11
2.5 Uninstall, Install a New Version . . . 12
2.6 Note about the Evaluation Version . . . 12
3 License Management . . . 13
3.1 Graphical License Manager Tool . . . 13
3.2 Command Line License Manager Tool . . . 14
3.3 License Key Storage . . . 14
4 Programming Interfaces . . . 15
4.1 Visual Basic 6 . . . 15
4.2 C/C++ . . . 17
4.3 .NET . . . 18
5 User's Guide . . . 20
5.1 Overview of the API . . . 20
5.2 How does the API work in general? . . . 21
5.3 Encryption . . . 21
5.4 Fonts . . . 23
5.5 Cryptographic Provider . . . 24
PKCS#11 Provider . . . 25
Microsoft CryptoAPI Provider . . . 26
3-Heights™ Signature Creation and Validation Service . . . 29
SwissSign Personal Signing Service . . . 31
SwissSign SuisseID Signing Service . . . 32
QuoVadis sealsign . . . 33
Swisscom All-in Signing Service . . . 34
Custom Signature Handler . . . 35
5.6 How to Create Digital Signatures . . . 35
How to Sign a PDF Document . . . 36
How to create a Preview of a signed Document . . . 36
How to Create a PAdES LTV Signature . . . 37
How to Create a Time-stamp Signature . . . 37
How to Create a Visual Appearance of a Signature . . . 37
Guidelines for Mass Signing . . . 38
Miscellaneous . . . 39
5.7 How to Validate Digital Signatures . . . 41
5.8 Advanced Guide . . . 44
5.9 Stamping . . . 45
6 Reference Manual . . . 51
6.1 PdfSecure Interface . . . 51
AddDocMDP Signature . . . 51
AddPreparedSignature . . . 52
AddSignature . . . 52
AddSignatureField . . . 52
AddStamps . . . 52

AddStampsMem . . . 52
AddTimeStampSignature . . . 53
BeginSession . . . 53
Close . . . 53
ErrorCode . . . 53
ErrorMessage . . . 53
EndSession . . . 53
ForceEncryption . . . 54
ForceSignature . . . 54
GetPDF . . . 54
GetRevison . . . 54
GetSignature . . . 54
InfoEntry . . . 55
NoCache . . . 55
Open . . . 55
OpenMem . . . 55
RevisionCount . . . 56
SaveAs . . . 56
SaveInMemory . . . 58
SetSessionProperty . . . 58
SignatureCount . . . 58
SignPreparedSignature . . . 59
SignSignatureField . . . 59
Terminate . . . 59
TestSession . . . 59
ValidateSignature . . . 60
6.2 PdfSignature Interface . . . 60
ContactInfo . . . 60
Date . . . 61
DocumentHasBeenModified . . . 61
Email . . . 61
EmbedRevocationInfo . . . 61
FillColor . . . 62
Filter . . . 62
FontName1 . . . 62
Fontname2 . . . 62
FontSize1 . . . 63
FontSize2 . . . 63
HasSignature . . . 63
ImageFileName . . . 63
Issuer . . . 63
LineWidth . . . 63
Location . . . 64
Name . . . 64
PageNo . . . 64
Provider . . . 64
ProxyURL . . . 65
ProxyCredentials . . . 65
Reason . . . 65
Rect . . . 65
Revision . . . 66
SerialNumber . . . 66
SignerFingerprint . . . 66
SignerFingerprintStr . . . 66
Store . . . 66
StoreLocation . . . 66
StrokeColor . . . 67
SubFilter . . . 67
Text1 . . . 67
Text2 . . . 67
TimeStampCredentials . . . 68
TimeStampFingerprint . . . 68
TimeStampURL . . . 68
6.3 Enumerations . . . 69
TPDFErrorCode . . . 69
TPDFPermission . . . 70

7 Licensing and Copyright 71
8 Contact 71

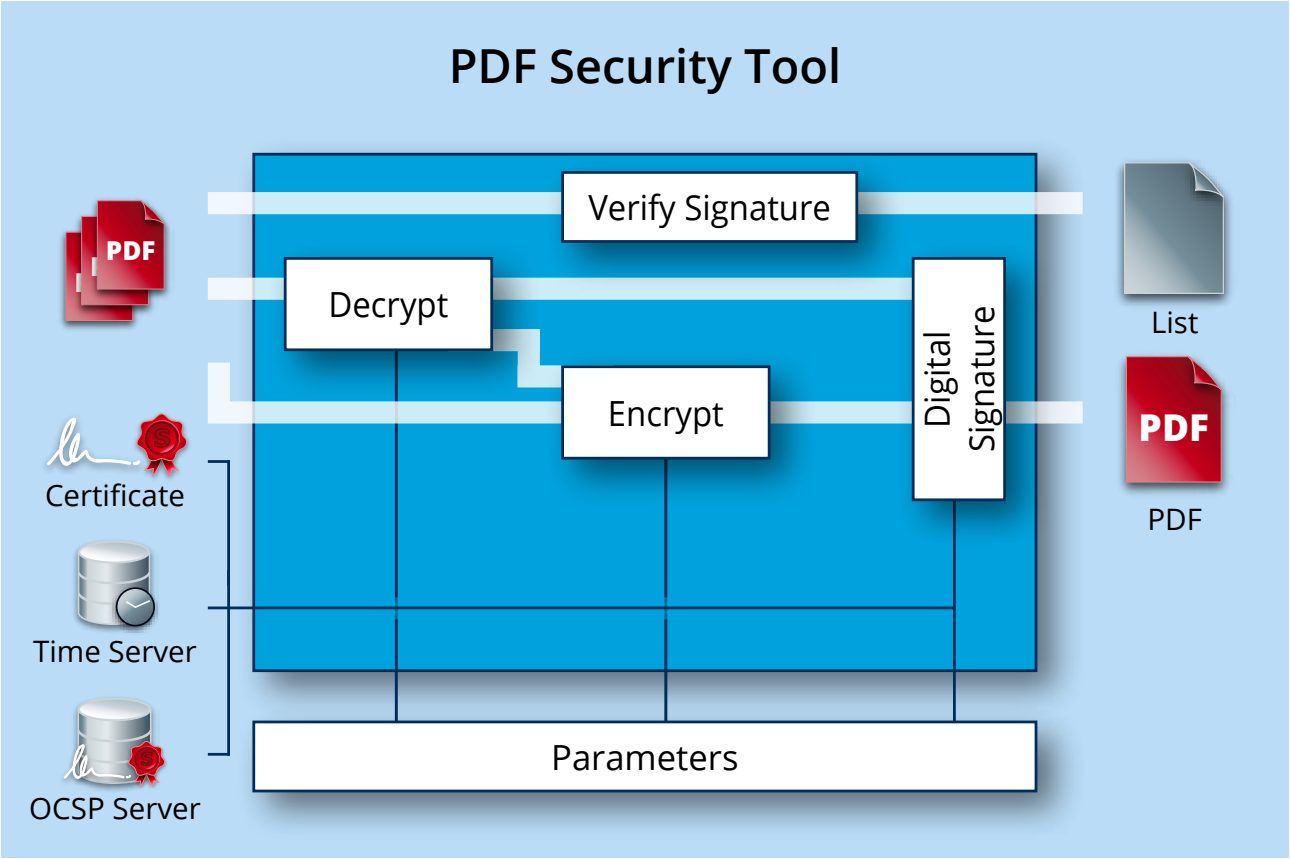
1 Introduction

1.1 Description

The 3-Heights™ PDF Security API enables the application of digital signatures to PDF documents and their subsequent protection through setting passwords and user authorizations.

Both standard signatures and qualified signatures that use signature cards (“smart cards”, “USB tokens”, “HSM”) can be used.

PDF documents used in professional circumstances contain important information that needs to be protected against misuse and unintentional alteration. This is achieved by protecting PDF documents through encryption and user authorization rights.



When exchanging electronic documents the ability to ascertain that a document is authentic and has not been manipulated on its way from sender to recipient is of particular importance. This is only achievable through the use of electronic signatures.

Through its interfaces (C, Java, .NET, COM) and thanks to its flexibility a developer can integrate the 3-Heights™ PDF Security API in virtually any application.

1.2 Functions

The 3-Heights™ PDF Security API enables users to encrypt and - if the passwords are known - decrypt PDF documents. The tool can set and cancel all known PDF user authorizations. It can, for instance, set an owner password so that only authorized users can edit and change the document. A user password ensures that only authorized users have access to the document's content. The tool's signature module allows the user to apply, read and verify both classic digital signatures and MDP (modification detection and prevention) signatures. The visibility and visual appearance of digital signatures can be adapted to suit requirements. The tool also supports customized signature handlers and types.

Features

- Apply simple, advanced and qualified electronic signatures
- Apply PAdES LTV (Long Term Validation) signatures
- Cache OCSP, CRL and TSP responses for mass signing
- Apply modification detection & prevention (MDP) signatures
- Apply document time-stamp signatures
- Encrypt and decrypt PDF documents
- Set user authorizations, including:
  - Print document
  - Modify document content
  - Extract or copy content
  - Add comments
  - Fill in form fields
  - Content extraction for accessibility
  - Assemble documents
  - Print in high resolution
- Set crypt and stream filters
- Set encryption strength
- Set owner and user password
- Stamping
  - PDF/A compliant stamps
  - Modify existing stamps
  - Stamping of signed documents preserves existing signatures

Formats

Input Formats:

- PDF 1.x (e.g. PDF 1.4, PDF 1.5)
- PDF/A-1, PDF/A-2, PDF/A-3

Target Formats:

- PDF 1.x (e.g. PDF 1.4, PDF 1.5)
- PDF/A-1, PDF/A-2, PDF/A-3

Compliance

Standards: ISO 32000 (PDF 1.7), ISO 19005-1 (PDF/A-1), ISO 19005-2 (PDF/A-2) , ISO 19005-3 (PDF/A-3), PAdES Part 2 and Part 4 (Long Term Validation, LTV)

1.3 Interfaces

The following interfaces are available: C, Java, .NET, COM.

1.4 Operating Systems

- Windows XP, Vista, 7, 8, 8.1 - 32 and 64 bit
- Windows Server 2003, 2008, 2008 R2, 2012, 2012 R2 - 32 and 64 bit
- HP-UX 11 and later PA-RISC2.0 32 bit or HP-UX 11i and later ia64 (Itanium) 64 bit
- IBM AIX 5.1 and later (64 bit)
- Linux (32 and 64 bit)
- Mac OS X 10.4 and later (32 and 64 bit)
- Sun Solaris 2.8 and later, SPARC and Intel
- FreeBSD 4.7 and later 32 bit or FreeBSD 9.3 and later 64 bit (on request)

1.5 How to Best Read this Manual

If you are reading this manual for the first time, i.e. would like to evaluate the software, the following steps are suggested.

1. Read the chapter Introduction to verify this product meets your requirements.
2. Identify what interface your programming language uses.
3. Read and follow the instructions in the chapter Installation And Deployment.
4. In the chapter Programming Interfaces find your programming language. Please note that not every language is covered in this manual.
   For many programming languages there is sample code available. For a start it is generally best to refer to these samples rather than writing code from scratch.
5. (Optional) Read the chapter User's Guide for general information about the API. Read Programmer's Reference for specific information about the functions of the API.

1.6 Digital Signatures

Overview

Digital signature is a large and slightly complex topic. This manual gives an introduction to digital signatures and describes how the 3-Heights™ PDF Security API is used to apply them. It does however not describe all the technical details.

Terminology

Digital Signature is a cryptographic technique of calculating a number (a digital signature) for a message. Creating a digital signature requires a private key from a certificate. Validating a digital signature and its authorship requires a public key. Digital Signature is a technical term.

Electronic Signature is a set of electronic data that is merged or linked to other electronic data in order to authenticate it. Electronic Signatures can be created by means of a digital signature or other techniques. Electronic Signature is a legal term.

Table: Abbreviations	
CA	Certification Authority
CMS	Cryptographic Message Syntax
CRL	Certificate Revocation List
CSP	Cryptographic Service Provider
HSM	Hardware Security Module

OCSP	Online Certificate Status Protocol
PKCS	Public Key Cryptography Standards
QES	Qualified Electronic Signature
TSA	Time-stamp Authority
TSP	Time-stamp Protocol

Why Digitally Signing?

The idea of applying a digital signature in PDF is very similar to a handwritten signature: A person reads a document and signs it with their name. In addition to the name, the signature can contain further optional information, such as the date and location. A valid electronic signature is a section of data that can be used to:

- Ensure the integrity of the document
- Authenticate the signer of the document
- Prove existence of file prior to date (time-stamp)

Digitally signing a document requires a certificate and its private key. How to access and use a certificate is described in the chapter Cryptographic Provider.

In a PDF document, a digital signature consists of two parts:

- *A PDF related part*: This part consists of the PDF objects required to embed the signature into the PDF document. This part depends on the signature type (Document Signature, MDP Signature, see table below). Information such as name of the signer, reason, date, location is stored here. The signature may optionally have a visual appearance on a page of the PDF document, which can contain text, graphics and images. This part of the signature is entirely created by the 3-Heights™ PDF Security API.
- *A cryptographic part*: A digital signature is based on a cryptographic checksum (hash value) calculated from the content of the document that is being signed. If the document is modified at a later time, the computed hash value is no longer correct and the signature becomes invalid, i.e. the validation will fail and will report that the document has been modified since the signature was applied. Only the owner of the certificate and its private key is able to sign the document. However, anybody can verify the signature with the public key contained in the certificate. This part of the signature requires a cryptographic provider for some cryptographic data and algorithms.

The 3-Heights™ PDF Security API supports the following types of digital signatures:

- *Document Signature*: Check the integrity of the signed part of the document and authenticate the signer's identity. One or more signatures can be applied. A signed document can be modified and saved by incremental update. The state of the document can be re-created as it existed at the time of signing.
- *MDP (Modification detection and prevention) Signature*: Enable detection of disallowed changes specified by the author. A document can contain only one MDP signature; it must be the first in the document. Other document signatures may be present.
- *Document Time-stamp Signature*: Establish the exact content of the file at the time indicated by the Time-stamp. One or more document Time-stamp signatures can be applied. A signed document can be modified and saved by incremental update.

1.6.1 What is an Electronic Signature?

There are different types of electronic signatures, which normally are defined by national laws, and therefore are different for different countries. The type of electronic signatures required in a certain process is usually defined by national laws. Quite advanced in this manner are German-speaking countries where such laws and an established terminology exist. The English terminology is basically a translation from German. Three types of electronic signatures are distinguished:

- Simple Electronic Signature "Einfache Elektronische Signatur"
- Advanced Electronic Signature "Fortgeschrittene Elektronische Signatur"
- Qualified Electronic Signature (QES) "Qualifizierte Elektronische Signatur"

All applied digital signatures are PDF/A and PAdES compliant.

Simple Electronic Signature

A simple electronic signature requires any certificate that can be used for digital signing. The easiest way to retrieve a certificate, which meets that requirement, is to create a so called self-signed certificate. Self-signed means it is signed by its owner, therefore the issuer of the certificate and the approver of the legitimacy of a document signed by this certificate is the same person.

Example: Anyone could create a self-signed certificate issued by "Peter Pan" and issued to "Peter Pan". Using this certificate one is able to sign in the name of "Peter Pan".

If a PDF document is signed with a simple electronic signature and the document is changed after the signature has been applied, the signature becomes invalid. However, the person who applied the changes, could at the same time (maliciously) also remove the existing simple electronic signature and - after the changes - apply a new, equally looking Simple Electronic Signature and falsify its date. As we can see, a simple electronic signature is neither strong enough to ensure the integrity of the document nor to authenticate the signer.

This drawback can be overcome using an Advanced or Qualified Electronic Signature.

Advanced Electronic Signature

Requirements for advanced certificates and signatures vary depending on the country where they are issued and used.

An advanced electronic signature is based on an advanced certificate that is issued by a recognized certificate authority (CA) in this country, such VeriSign, SwissSign, QuoVadis. In order to receive an advanced certificate, its owner must prove his identity, e.g. by physically visiting the CA and presenting his passport. The owner can be an individual, a legal person or another entity.

An advanced certificate contains the name of the owner, the name of the CA, its period of validity and other information.

The private key of the certificate is protected by a PIN, which is only known to its owner.

This brings the following advantages over a simple electronic signature:

- The signature authenticates the signer.
- The signature ensures the integrity of the signed content.

Qualified Electronic Signature

Requirements for qualified certificates and signatures vary depending on the country where they are issued and used.

A Qualified Electronic Signature is similar to an advanced electronic signature, but has higher requirements. The main differences are:

- It is based on a qualified certificate, which is provided as a hardware token (USB stick, smart card).
- For every signature it is required to enter the PIN code manually. This means that only one signature can be applied at a time.
- Certificate revocation information (OCSP/CRL) can be acquired from an online service. The response (valid, revoked, etc.) must be embedded in the signature.
- A time-stamp (TSP) that is acquired from a trusted time server (TSA) may be required.

This brings the following advantages over an advanced electronic signature:

- The signature ensures the certificate was valid at the time when the document was signed (due to the embedding of the OCSP/CRL response).
- The signature ensures the integrity of the time of signing (due to the embedding of the time-stamp).
- Legal processes that require a QES are supported.

Note that a Time-stamp can be added to any type of signature. OCSP/CRL responses are also available for some advanced certificates.

1.6.2 How to Create Electronic Signatures

Preparation Steps

1. Identify whether an advanced or a qualified signature is required. For most automated processes an advanced signature is sufficient.
2. Acquire a corresponding certificate from a CA. Note that some CA offer USB sticks or smart cards that contain both, an advanced and a qualified certificate.
3. Setup and configure the certificate's Cryptographic Provider
   - In case the certificate resides on hardware such as an USB token or a Smart Card, the required middleware (driver) needs to be installed.
   - In case the certificate is a soft certificate, it must be imported into the certificate store of a cryptographic provider.
4. Optional: Acquire access to a trusted time server (TSA) (e.g. from the CA of your signing certificate).
5. Apply the signature by providing the following information:
   - Values for the selection of the signing certificate (e.g. the name of the certificate)
   - The Cryptographic Provider where the certificate is located
   - Optional: Time-stamp service URL (e.g. "http://server.mydomain.com:80/tsa")
   - Optional: Time-stamp service credentials (e.g. username:password)
   - Optional: Embed revocation information (default: true)
   - Optional: Visual appearance of the signature on a page of the document (e.g. an image).

Example: Steps to Add an Electronic Signature

The 3-Heights™ PDF Security API applies PDF/A compliant signatures. This means if a PDF/A document is digitally signed, it remains PDF/A compliant.

In order to add an electronic signature with the 3-Heights™ PDF Security API the following steps need to be done:

1. Create a new PdfSignature object
2. As value of the PdfSignature's name, the name of the certificate that is to be used must be provided. The name of the certificate corresponds to the value "Issued to:".
3. If the certificate's private key is PIN protected, the PIN can be passed in the provider configuration.
4. Additional parameters can now be set such as the reason why the signature is applied, etc.

In Visual Basic the four steps above look like this:

```
Dim Document As New PDFSECUREAPILib.PdfSecure
Document.Open "input.pdf", ""

Dim Signature As New PDFSECUREAPILib.PdfSignature
Signature.Name = "Philip Renggli"
Signature.Provider = "cvp11.dll;0;secret-pin"
Signature.Reason = "I reviewed the document"                  ' optional
Signature.TimeStampURL = "http://server.mydomain.com:80/tsa" ' optional

Document.AddSignature Signature
Document.SaveAs "output.pdf"
Document.Close
```

Note: On Non-Windows platforms additional settings are required to be set, such as fonts.

The visual appearance of the digital signature on a page of the resulting output-document looks as shown below:

2 Installation and Deployment

2.1 Windows

The retail version of the 3-Heights™ PDF Security API comes as a ZIP archive containing various files including runtime binary executable code, files required for the developer, documentation and license terms.

1. Download the ZIP archive of the product from your download account at http://www.pdf-tools.com.
2. Unzip the file using a tool like WinZip available from WinZip Computing, Inc. at http://www.winzip.com to a directory on your hard disk where your program files reside (e.g. *C:\Program Files\PDF Tools AG*).
3. Check the appropriate option to preserve file paths (folder names). The unzip process now creates the following subdirectories:

   bin: Contains the runtime executable binary code.

   doc: Contains documentation files.

   include: Contains header files to include in your C / C++ project.

   samples: Contains sample programs in various programming languages.

   There is the option to download the software as MSI file, which makes the installation easier.
4. Optionally register your license key using the License Manager.
5. Identify which interface you are using. Perform the specific installation steps for that interface described in chapter Interfaces.
6. If you want to sign documents, proceed with setting up your cryptographic provider as described in chapter Cryptographic Provider.
7. If you want to stamp text, proceed with setting the fonts required as described in chapter Fonts.

2.2 Unix

This section describes installation steps required on all Unix platforms, which includes Linux, Mac OS X, Sun Solaris, IBM AIX, HP-UX, FreeBSD and others.

The Unix version of the 3-Heights™ PDF Security API provides two interfaces:

- Java interface
- Native C interface

Here is an overview of the shared libraries and other files that come with the 3-Heights™ PDF Security API:

Table: File Description

Name	Description
`bin/libPdfSecureAPI.so`	This is the shared library that contains the main functionality. The file extension varies depending on the UNIX platform.
`doc/*.*`	Documentation
`bin/SECA.jar`	Java API archive.
`include/*.h`	Contains files to include in your C/C++ Project.

Example code written in different programming languages are available at product page of the PDF Tools AG website (http://www.pdf-tools.com).

All Unix Platforms

1. Unpack the archive in an installation directory, e.g. */opt/pdf-tools.com/*
2. Copy or link the shared object into one of the standard library directories, e.g:
   `ln -s /opt/pdf-tools.com/bin/libPdfSecureAPI.so /usr/lib`
3. Verify that the GNU shared libraries required by the product are available on your system:
   - *On Linux*: `ldd libPdfSecureAPI.so`
   - *On AIX*: `dump -H libPdfSecureAPI.so`

   In case you have not installed the GNU shared libraries yet, proceed as follows:
   (a) Go to http://www.pdf-tools.com and navigate to "Support" → "Resouces".
   (b) Download the GNU shared libraries for your platform.
   (c) Extract the archive and copy or link the libraries into your library directory, e.g */usr/lib* or */usr/lib64*.
   (d) Verify that the GNU shared libraries required by the product are available on your system now.
4. Optionally register your license key using the Command Line License Manager Tool.
5. Identify which interface you are using. Perform the specific installation steps for that interface described in chapter Interfaces.
6. If you want to sign documents, proceed with setting up your cryptographic provider as described in chapter Cryptographic Provider.
7. If you want to stamp text, proceed with setting the fonts required as described in chapter Fonts.

MAC OS/X

The shared library must have the extension .jnilib for use with Java. We suggest that you create a file link for this purpose by using the following command:

```
ln libPdfSecureAPI.dylib libPdfSecureAPI.jnilib
```

2.3 Interfaces

The 3-Heights™ PDF Security API provides four different interfaces. The installation and deployment of the software depend on the interface you are using. The table below shows the supported interfaces and examples with which programming languages they can be used.

Table: Interfaces

Interface	Programming Languages
`.NET`	The MS software platform .NET can be used with any .NET capable programming language such as: ▪ C# ▪ VB .NET ▪ J# ▪ others This interface is available in the Windows version only.
`JNI`	The Java native interface (JNI) is for use with Java.
`COM`	The component object model (COM) interface can be used with any COM-capable programming language, such as: ▪ MS Visual Basic ▪ MS Office Products such as Access or Excel (VBA) ▪ C++ ▪ VBScript ▪ others This interface is available in the Windows version only.
`C`	The native C interface is for use with C and C++.

Development

The software developer kit (SDK) contains all files that are used for developing the software. The role of each file with respect to the four different interfaces is shown in Table: Files for Development. The files are split in four categories:

Req. This file is required for this interface.

Opt. This file is optional (e.g. *Inet.dll* is used for http: and other connections. When using the API locally, this file is not used). See also Table: File Description to identify which files are required for your application.

Doc. This file is for documentation only. An empty field indicates this file is not used at all for this particular interface.

Table: Files for Development

Name	.NET	JNI	COM	C
`bin\PdfSecureAPI.dll`	**Req.**	**Req.**	**Req.**	**Req.**
`bin\pdcjk.dll`	Opt.	Opt.	Opt.	Opt.
`bin\*NET.dll`	**Req.**			
`bin\*NET.xml`	Doc.			
`doc\*.pdf`	Doc.	Doc.	Doc.	Doc.
`doc\PdfSecureAPI.idl`			Doc.	

doc\javadoc*.*		Doc.		
include\pdfsecureapi_c.h				**Req.**
include*.*				Opt.
jar\SECA.jar		**Req.**		
lib\PdfSecureAPI.lib				**Req.**
samples*.*	Doc.	Doc.	Doc.	Doc.

The purpose of the most important distributed files of is described in Table: File Description.

Table: File Description

Name	Description
bin\PdfSecureAPI.dll	This is the DLL that contains the main functionality (required).
bin\pdcjk.dll	This DLL contains support for Asian languages. It is loaded from the module path.
bin*NET.dll	The .NET assemblies are required when using the .NET interface. The files *bin*NET.xml* contain the corresponding XML documentation for MS Studio.
include\pdferror.h	Supplementary C header file containing error codes.
doc*.*	Various documentations.
include*.*	Contains files to include in your C / C++ project.
jar\SECA.jar	The Java wrapper.
lib\PdfSecureAPI.lib	The Object File Library needs to be linked to the C/C++ project.
samples*.*	Contains sample programs in different programming languages.

Deployment

For the deployment of the software only a subset of the files are required. Which files are required (**Req.**), optional (Opt.) or not used (empty field) for the four different interfaces is shown in the table below.

Table: Files for Deployment

Name	.NET	JNI	COM	C
bin\PdfSecureAPI.dll	**Req.**	**Req.**	**Req.**	**Req.**
bin\pdcjk.dll	Opt.	Opt.	Opt.	Opt.
bin*NET.dll	**Req.**			
jar\SECA.jar		**Req.**		

The deployment of an application works as described below:

1. Identify the required files from your developed application (this may also include color profiles)
2. Identify all files that are required by your developed application
3. Include all these files into an installation routine such as an MSI file or simple batch script
4. Perform any interface-specific actions (e.g. registering when using the COM interface)

Example:

This is a very simple example of how a COM application written in Visual Basic 6 could be deployed.

1. The developed and compiled application consists of the file securer.exe. Color profiles are not used.
2. The application uses the COM interface and is distributed on Windows only.
   - The main DLL *PdfSecureAPI.dll* must be distributed.
   - Asian text should be supported, thus *pdcjk.dll* is distributed.
3. All file are copied to the target location using a batch script. This script contains the following commands:
   ```
   COPY PdfSecureAPI.dll %targetlocation%\.
   COPY pdcjk.dll %targetlocation%\.
   ```
4. For COM, the main DLL needs to be registered in silent mode (/s) on the target system. This step requires Power-User privileges and is added to the batch script.
   ```
   REGSVR32 /s %targetlocation%\PdfSecureAPI.dll
   ```

2.4 Interface Specific Installation Steps

COM Interface

Registration: Before you can use the 3-Heights™ PDF Security API component in your COM application program you have to register the component using the *regsvr32.exe* program that is provided with the Windows operating system. The following command shows the registration of *PdfSecureAPI.dll*. Note that in Windows Vista and later, the command needs to be executed from an administrator shell.

```
regsvr32 C:\Program Files\PDF Tools AG\bin\PdfSecureAPI.dll
```

If you are using a 64 bit operating system and would like to register the 32 bit version of the 3-Heights™ PDF Security API, you need to use the regsvr32 from the directory `%SystemRoot%\SysWOW64` instead of `%SystemRoot%\System32`.

If the registration process succeeds, a corresponding dialog window is displayed. The registration can also be done silently (e.g. for deployment) using the switch `/s`.

Other Files: The other DLLs do not need to be registered, but for simplicity it is suggested that they reside in the same directory as the *PdfSecureAPI.dll*.

Java Interface

For compilation and execution: When using the Java interface, the Java-wrapper *jar\SECA.jar* needs to be on the CLASSPATH. This can be done by either adding it to the environment variable CLASSPATH, or by specifying it using the switch -classpath

```
javac -classpath .;C:\pdf-tools\jar\SECA.jar sample.java
```

For execution: Additionally the library *PdfSecureAPI.dll* needs to be on the Java system property `java.library.path`. This can be achieved by either adding it dynamically at program startup before using the API, or by specifying it using the switch `-Djava.library.path` when starting the Java VM.

```
java -classpath .;C:\pdf-tools\jar\SECA.jar -Djava.library.path=.;C:\pdf-tools\bin
    sample
```

.NET Interface

The 3-Heights™ PDF Security API does not provide a pure .NET solution. Instead, it consists of .NET assemblies, which are added to the project and a native DLL, which is called by the .NET assemblies. This has to be accounted for when installing and deploying the tool.

The .NET assemblies (**NET.dll*) are to be added as references to the project. They are required at compilation time. See also chapter "Getting Started".

PdfSecureAPI.dll is not a .NET assembly, but a native DLL. It is not to be added as a reference in the project.

The native DLL *PdfSecureAPI.dll* is called by the .NET assembly *PdfSecureNET.dll*.

PdfSecureAPI.dll must be found at execution time by the Windows operating system. The common way to do this is adding *PdfSecureAPI.dll* as an existing item to the project and set its property "Copy to output directory" to "Copy if newer".

Alternatively the directory where *PdfSecureAPI.dll* resides can be added to the environment variable "PATH" or it can simply be copied manually to the output directory.

C Interface

- The header file *pdfsecureapi_c.h* needs to be included in the C/C++ program.
- The library *PdfSecureAPI.lib* needs to be linked to the project.
- The dynamic link library *PdfSecureAPI.dll* needs to be in path of executables (e.g. on the environment variable "PATH").

2.5 Uninstall, Install a New Version

If you used the MSI for the installation, go to Start ->3-Heights™ PDF Security API ... ->Uninstall...
If you used the ZIP file: In order to uninstall the product undo all the steps done during installation, e.g. un-register using regsvr32 -u, delete all files, etc.

Installing a new version does not require to previously uninstall the old version. The files of the old version can directly be overwritten with the new version. If using the COM interface, the new DLL must be registered, un-registering the old version is not required.

2.6 Note about the Evaluation Version

The evaluation versions of the 3-Heights™ products automatically add a watermark to the output files.

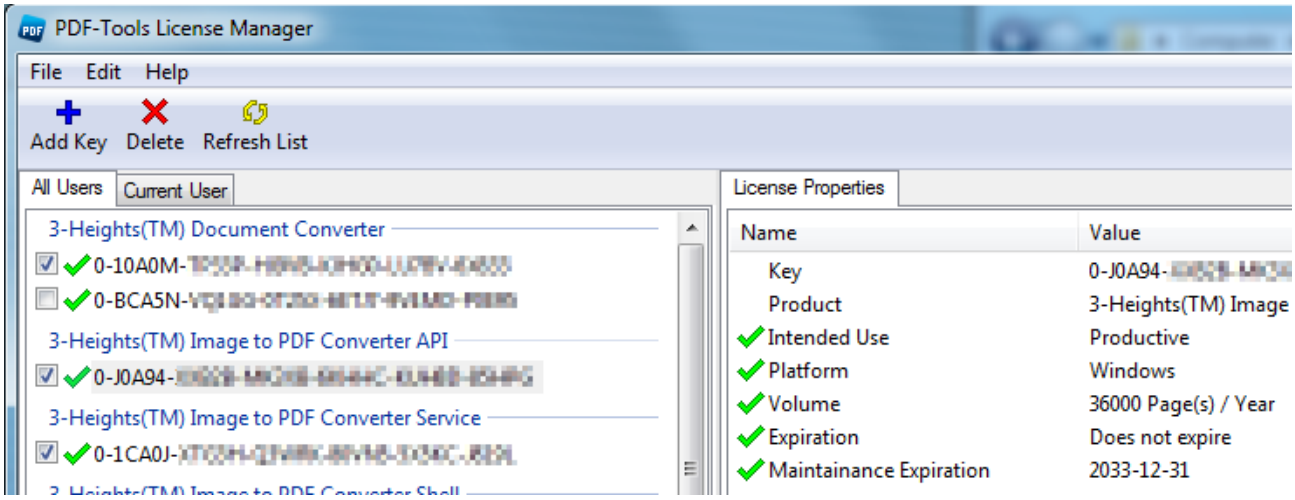
3 License Management

There are three possibilities to pass the license key to the application:

1. The license key is installed using the GUI tool (Graphical user interface). This is the easiest way if the licenses are managed manually. It is only available on Windows.
2. The license key is installed using the shell tool. This is the preferred solution for all non-Windows systems and for automated license management.
3. The license key is passed to the application at runtime via the “LicenseKey” property. This is the preferred solution for OEM scenarios.

3.1 Graphical License Manager Tool

The GUI tool *LicenseManager.exe* is located in the *bin* directory of the product kit.



List all installed license keys

The license manager always shows a list of all installed license keys in the left pane of the window. This includes licenses of other PDF Tools products. The user can choose between:

- Licenses available for all users. Administrator rights are needed for modifications.
- Licenses available for the current user only.

Add and delete license keys

License keys can be added or deleted with the “Add Key” and “Delete” buttons in the toolbar.

- The “Add key” button installs the license key into the currently selected list.
- The “Delete” button deletes the currently selected license keys.

Display the properties of a license

If a license is selected in the license list, its properties are displayed in the right pane of the window.

Select between different license keys for a single product

More than one license key can be installed for a specific product. The checkbox on the left side in the license list marks the currently active license key.

3.2 Command Line License Manager Tool

The command line license manager tool `licmgr` is available in the bin directory for all platforms except Windows.

A complete description of all commands and options can be obtained by running the program without parameters:

```
licmgr
```

List all installed license keys

```
licmgr list
```

Add and delete license keys

Install new license key:

```
licmgr store X-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX
```

Delete old license key:

```
licmgr delete X-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX
```

Both commands have the optional argument `-s` that defines the scope of the action:

- `g`: For all users
- `u`: Current user

Select between different license keys for a single product

```
licmgr select X-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX
```

3.3 License Key Storage

Depending on the platform the license management system uses different stores for the license keys.

Windows

The license keys are stored in the registry:

- `HKLM\Software\PDF Tools AG` (for all users)
- `HKCU\Software\PDF Tools AG` (for the current user)

Mac OS X

The license keys are stored in the file system:

- `/Library/Application Support/PDF Tools AG` (for all users)
- `~/Library/Application Support/PDF Tools AG` (for the current user)

Unix/Linux

The license keys are stored in the file system:

- `/etc/opt/pdf-tools` (for all users)
- `~/.pdf-tools` (for the current user)

Note: The user, group and permissions of those directories are set explicitly by the license manager tool. It may be necessary to change permissions to make the licenses readable for all users. Example:

```
chmod -R go+rx /etc/opt/pdf-tools
```

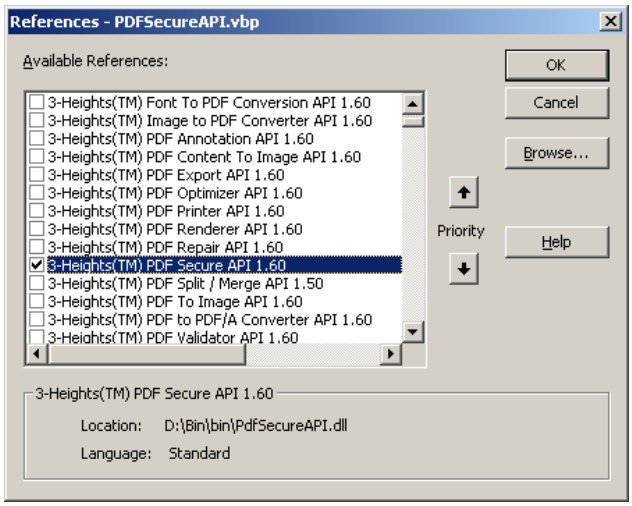
4 Programming Interfaces

4.1 Visual Basic 6

After installing the 3-Heights™ PDF Security API and registering the COM interface (see chapter Download and Installation), you find a Visual Basic 6 example *PdfSecureAPI.vbp* in the directory *samples/VB/*. You can either use this sample as a base for an application, or you can start from scratch.

If you start from scratch, here is a quick start guide for you:

1. First create a new Standard-Exe Visual Basic 6 project. Then include the 3-Heights™ PDF Security API component to your project.


References - PDFSecureAPI.vbp
Available References:
OK
Cancel
Browse...
Priority
Help
3-Heights(TM) Font To PDF Conversion API 1.60
3-Heights(TM) Image to PDF Converter API 1.60
3-Heights(TM) PDF Annotation API 1.60
3-Heights(TM) PDF Content To Image API 1.60
3-Heights(TM) PDF Export API 1.60
3-Heights(TM) PDF Optimizer API 1.60
3-Heights(TM) PDF Printer API 1.60
3-Heights(TM) PDF Renderer API 1.60
3-Heights(TM) PDF Repair API 1.60
3-Heights(TM) PDF Secure API 1.60
3-Heights(TM) PDF Split / Merge API 1.50
3-Heights(TM) PDF To Image API 1.60
3-Heights(TM) PDF to PDF/A Converter API 1.60
3-Heights(TM) PDF Validator API 1.60
3-Heights(TM) PDF Secure API 1.60
Location: D:\Bin\bin\PdfSecureAPI.dll
Language: Standard

2. Draw a new Command Button and optionally rename it if you like.
3. Double-click the command button and insert the few lines of code below. All that you need to change is the path of the file name.

Example:

```
Private Sub Command1_Click()
   Dim Secure As New PDFSECUREAPILib.PdfSecure
   Secure.Open "C:\input.pdf", ""
   Secure.SaveAs "C:\output.pdf", "", "pwd", ePermPrint, 40
   Secure.Close
End Sub
```

And that's all - four lines of code. Create the object, open the input file, create the output file with no user password, owner password "owner", allow printing and use 40 bit encryption key.

Example: More advanced

The following Visual Basic 6 sample assumes an interface with:

- Text fields (txt*) for the input and output file names, as well as the passwords.
- Check boxes (chk*) with a value to be set to 0 or 1 for all the permission flags.

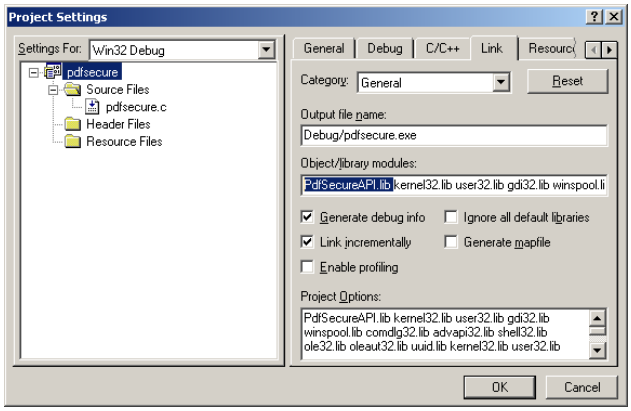
```
Private Sub CreateOutput_Click()
   Dim doc As New PDFSECUREAPILib.PdfSecure
   Dim iPerm As Integer
   done = doc.Open(txtInput.Text, txtPwd.Text)
   ' Open the input file
   If Not done Then
      If doc.ErrorCode = PDF_E_PASSWORD Then
         MsgBox "Input file is encrypted and Password not correct."
      Else
```

```
        MsgBox "Couldn't open input file."
      End If
      Exit Sub
   End If
   ' Set the permissions
   iPerm = chkPrint.Value * ePermPrint _
       + chkModify.Value * ePermModify _
       + chkCopy.Value * ePermCopy _
       + chkAnnot.Value * ePermAnnotate _
       + chkFill.Value * ePermFillForms _
       + chkExtr.Value * ePermSupportDisabilities _
       + chkAssemble.Value * ePermAssemble _
       + chkDPrint * ePermDigitalPrint
   iKey = 128
   ' Save the output file
   If Not doc.SaveAs(txtOutput.Text, txtUser.Text, txtOwner.Text, _ iPerm, iKey)
       Then
       MsgBox "Output file could not be created."
   End If
   done = doc.Close
End Sub
```

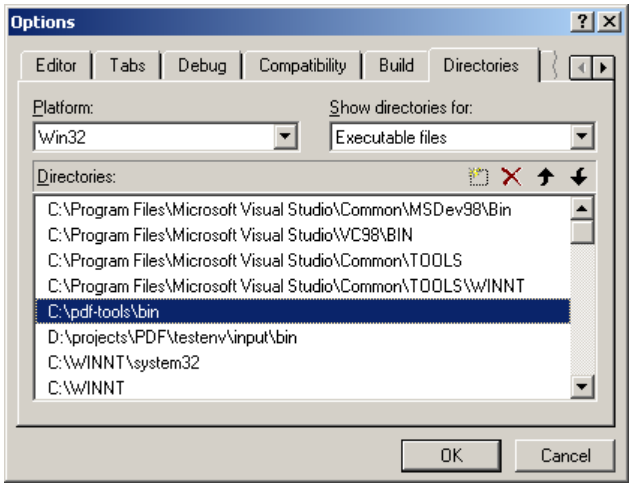
4.2 C/C++

In order to use the 3-Heights™ PDF Security API in a C project the following steps should be done. (Note: Steps and Screenshots are specifically described for the MS Studio 6)

1. Add the header files *PdfSecureapi_c.h* and pdfsecuritydecl.h to the include directories
2. Link to the object file library (Windows: *PdfSecureAPI.lib*)



3. Add the path where the dynamic link library *pdfsecureapi.dll* resides to the Executable files directories. E.g. as shown in the screenshot below. In most cases it works to simply add it to the environment variable PATH.



There is a C sample available within the ZIP archive of the evaluation and release version that shows how to decrypt and encrypt a PDF document, as well as how to add a digital signature. The C sample below is much simpler and does not add a digital signature. Before the C interface can be used to create objects, it must be initialized once. This is done using PdfSecureInitialize, to un-initialize use PdfSecureUnInitialize. Other than that, equal call sequences as in the COM interface can be used.

```
#include <stdio.h>
#include "pdfsecureapi_c.h"
#include "pdfsecuritydecl.h"

int main(int argc, char* argv[])
{
    TPdfSecure* pPdfSecure;
    PdfSecureInitialize();

    pPdfSecure = PdfSecureCreateObject();
    PdfSecureOpen(pPdfSecure, argv[1], "");
    PdfSecureSaveAsA(pPdfSecure, argv[2], "", "pwd", ePermPrint, 128, "", "");
    PdfSecureClose(pPdfSecure);

    PdfSecureDestroyObject(pPdfSecure);
```

```
    PdfSecureUnInitialize();
    return 0;
}
```

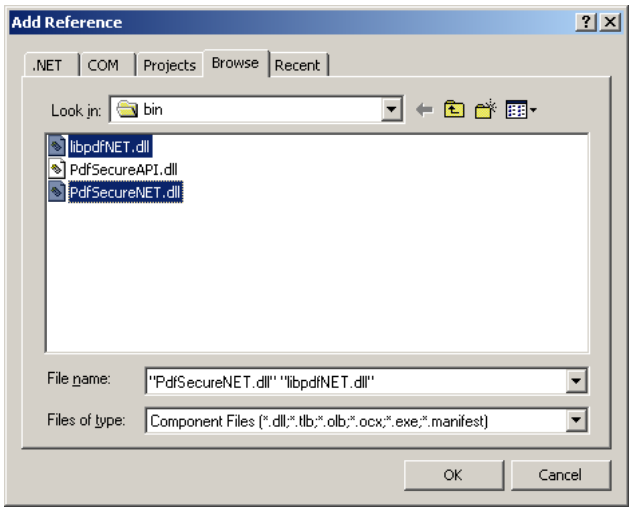
4.3 .NET

As opposed to previous versions, the Windows build numbers 1.7.1.* and later provide a .NET interface.

There should be at least one .NET sample for MS Visual Studio 2005 available in the ZIP archive of the Windows Version of the 3-Heights™ PDF Security API. The easiest for a quick start is to refer to this sample.

In order to create a new project from scratch, do the following steps:

1. Start Visual Studio and create a new C# or VB project.
2. Add a reference to the .NET assemblies. To do so, in the “Solution Explorer” right-click your project and select “Add Reference...”. The “Add Reference” dialog will appear. In the tab “Browse”, browse for the .NET assemblies *libpdfNET.dll* and *PdfSecureNET.dll*
   Add them to the project as shown below:

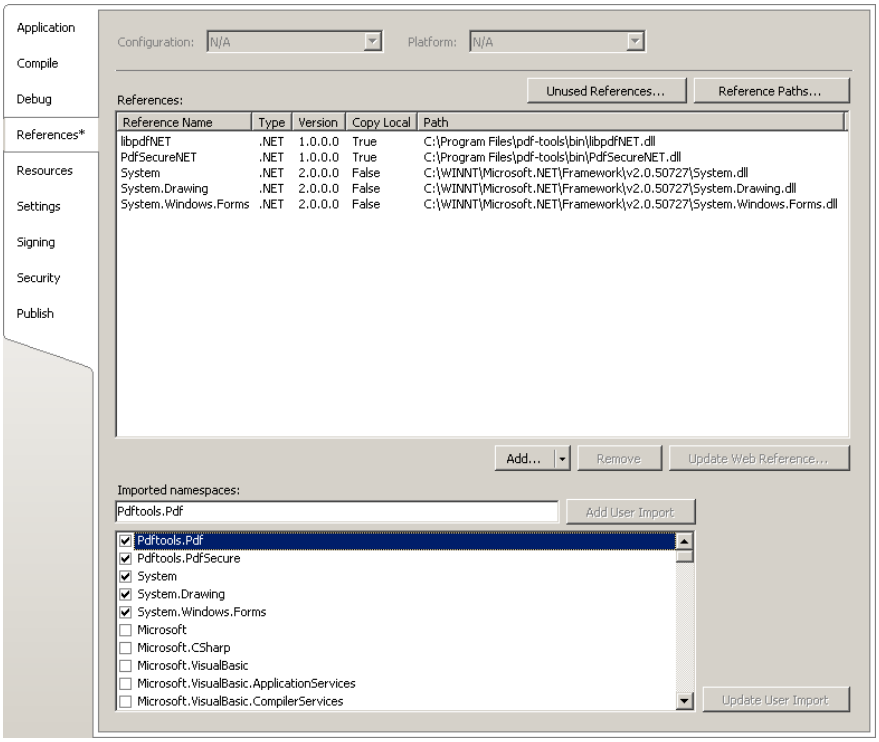


3. import namespaces (Note: This step is optional, but useful.)
4. Write Code

Steps 3 and 4 are shown separately for C# and Visual Basic.

Visual Basic

3. Double-click “My Project” to view its properties. On the left hand side, select the menu “References”. The .NET assemblies you added before should show up in the upper window.
   In the lower window import the namespaces *Pdftools.Pdf* and *Pdftools.PdfSecure*.
   You should now have settings similar as in the screenshot below:



4. The .NET interface can now be used as shown below:

```
Dim doc As New PdfSecure.Secure
Dim sig As New PdfSecure.Signature
doc.Open(...)
...
If Not doc.SaveAs("C:\temp\output.pdf", _
         "", _
         "pwd", _
         PDFPermission.ePermPrint, _
         128, _
         "V2", _
         "V2") = True Then
```

C#

3. Add the following namespaces:

```
using Pdftools.Pdf;
using Pdftools.PdfSecure;
```

4. The .NET interface can now be used as shown below:

```
Secure doc = new Secure();
Signature sig = new Signature();
doc.Open(...)
...
```

Deploying in .NET

When deploying a .NET solution, please refer to the following FAQ “Deploying in .NET”:
http://www.pdf-tools.com/pdf/Support/FAQ/Article.aspx?name=Deployment-In-NET

Troubleshooting: TypeInitializationException

The most common issue when using the .NET interface is if the native DLL is not found at execution time. This normally manifests when the constructor is called for the first time and exception is thrown - normally of type System.TypeInitializationException.

To resolve that ensure the native DLL is found at execution time. For this, see section .NET Interface in the chapter Installation or the following FAQ:
https://www.pdf-tools.com/pdf/Support/FAQ/Article.aspx?name=Exception-type-initializer

5 User’s Guide

5.1 Overview of the API

What is the 3-Heights™ PDF Security API about?

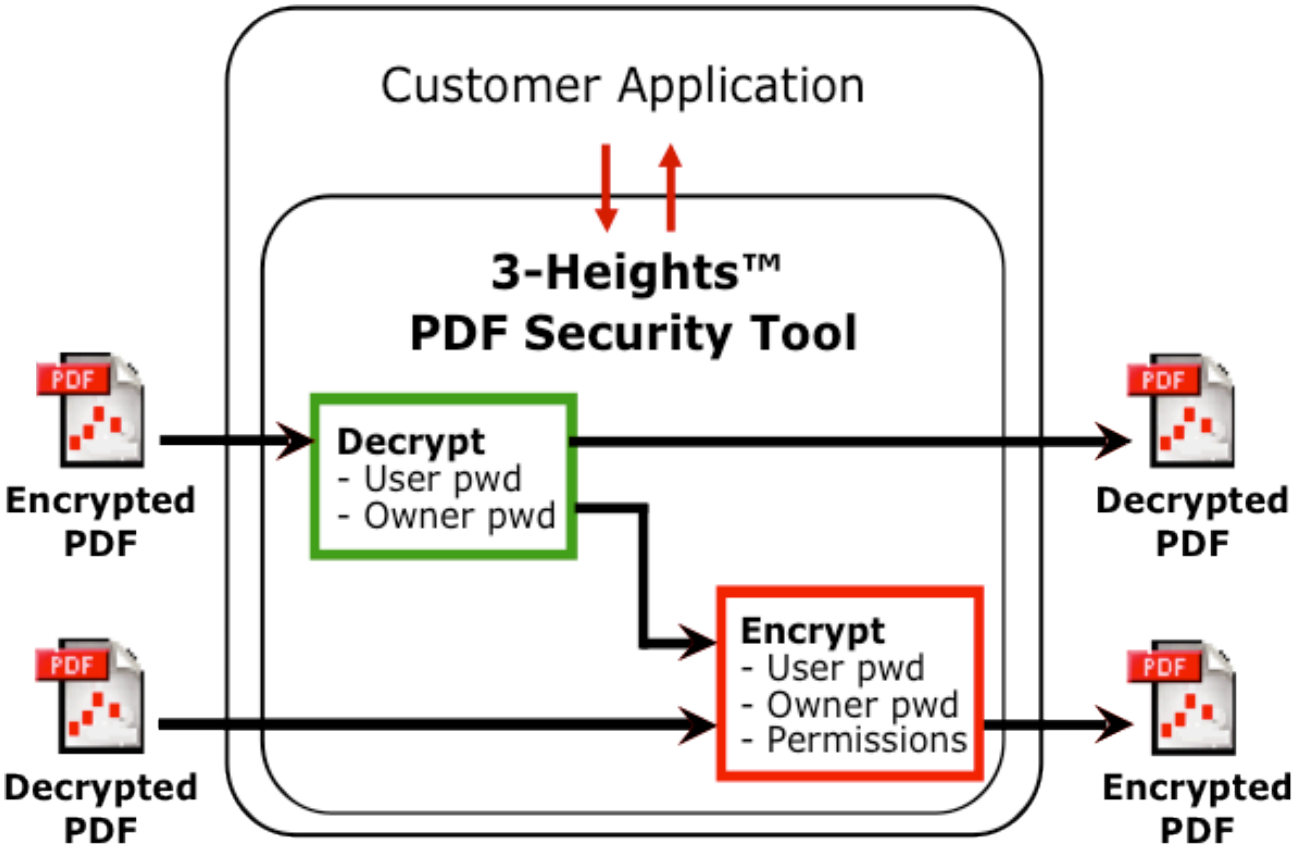
The 3-Heights™ PDF Security API provides three key functionalities related to security in PDF documents:

1. Deal with encryption, decryption and access permissions of PDF documents
2. Deal with digital signatures
3. Apply stamps to PDF documents

These three functionalities can be combined; they however are not closely related. What encryption and what a digital signature is, is described in the upcoming chapters.

5.2 How does the API work in general?

The 3-Heights™ PDF Security API requires a PDF document as input. In this manual, that document is referred to as input-document. In the graphic below that's the document on the left hand side. The document can be opened from file or from memory. If the document is encrypted, it is in a first step decrypted.



In the next step, application specific operations are applied. These can be setting new passwords and access permissions or add a digital signature (not shown in graphic).

After that, a new PDF document is created according to the defined settings. In this manual, the new resulting document is referred to as output-document. The input-document is never changed by the 3-Heights™ PDF Security API. Thus, the output-document must be a new document. It is not possible to directly overwrite the input-document.

5.3 Encryption

Encryption and how it works in PDF

A PDF document can be encrypted to protect its contents from unauthorized access. The encryption process applies encryption to all streams (e.g. images) and strings, but not to other items in the PDF document. This means the structure of the PDF document is accessible, but the content of its pages is encrypted.

When encryption is used in PDF, a security handler must be selected. The 3-Heights™ PDF Security API always uses the standard security handler which, according to the PDF Specification, has to be supported by any software that can process encrypted PDF documents.

For more detailed information about PDF encryption in general, see PDF Reference, chapter 3.5.

Owner Password and User Password

The standard security handler allows access permissions and up to two passwords to be specified for a document: An owner password and a user password.

The *user password* protects the document against unauthorized opening and reading. If a PDF document is protected by a user password, either the user or owner password must be provided to open and read the document. If a document has a user password, it must have an owner password as well. If no owner password is defined, the owner password is the same as the user password.

The *owner password* is also referred to as the author's password. This password grants full access to the document. Not only can the document be opened and read, it also allows for changing the document's security settings (access permission and passwords).

The following table shows the four possible combinations of passwords and how an application processing such a PDF document behaves.

Table: Owner and User Passwords		
UserPwd	**OwnerPwd**	**Behavior**
`none`	`none`	Everyone can read. Everyone can change security settings. (No encryption)
`none`	`set`	Everyone can read. The user password is an empty string. Owner password required to change security settings.
`set`	`none`	User password required to read. The owner password is equal to the user password. User password required to change security settings.
`set`	`set`	User or owner password required to read. Owner password required to change security settings.

Permission Flags

What operations in a PDF document are granted is controlled via its permission flags. In order to set permission flags, the PDF document must be encrypted and have an owner password. The owner password is required to initially set or later change the permission flags.

These access permission flags are:

- Modifying the content of the document
- Copying or extracting text and graphics from the document
- Adding or modifying text annotations and interactive form fields
- Printing the document (low or high quality)
- Filling in form and digitally signing the document
- Assembling the document (inserting, rotating, deleting pages, etc.)

How to Encrypt a PDF Document

If either of the passwords or permission flags is set, the document is encrypted.

If only a user password is set, but no owner password and no permission flags, the owner password is equal to the user password and all permissions are granted.

In the 3-Heights™ PDF Security API, the passwords and permission flags are provided as parameters of the SaveAs function. Note that the PDF Specification accepts an empty string as password. PDF applications by default try to open documents with the empty string password.

To encrypt a document and protect it against any manipulations other than printing, the document must have an owner password and the print permission flag set. In Visual Basic such as SaveAs call would look like this:

```
SaveAs("C:\temp\output.pdf", "", "ownerpwd", ePermPrint)
```

To encrypt a document similar as above, but in addition also have the application prompt the user for a password to open and read the document, you can add a user password as additional parameter in the SaveAs function:

```
SaveAs("C:\temp\output.pdf", "userpwd", "ownerpwd", ePermPrint)
```

To not encrypt a document at all, set empty passwords and ePermNoEncryption (-1) for permission flags:

```
SaveAs("C:\temp\output.pdf", "", "", ePermNoEncryption)
```

How to Read an Encrypted PDF Document

A PDF document which is not encrypted or protected with an owner password only, can be read and decrypted by the 3-Heights™ PDF Security API's Open function without providing a password.

In Visual Basic that looks like this: `Open("C:\temp\input.pdf", "")`

A PDF document which is protected by a user password can only be opened if either the user or the owner password is provided as parameter in the Open function. Technically it does not matter later on which of the two passwords was provided, because both will grant full access to the document. However it is up to the application programmer to distinguish between input-documents that are password protected or not.

How secure is PDF Encryption?

Any PDF application that is to process or display a PDF document must be able to read and decrypt the contents of the pages in order to be able to display them. It technically cannot display an encrypted text or image without first decrypting it. A PDF application program has therefore full access to any PDF document it can decrypt and display.

PDF application programs, such as all products of the PDF Security API family, or Adobe Acrobat, can open and decrypt PDF documents which have an owner password but no user password, without knowing that password. Otherwise they couldn't display the document. The application at that point has full access to the document. However this does not imply the user of this application is given the same access rights. The user should only be given the access permissions defined by the permission flags and the password he provided. Any PDF application which behaves different from that can allow for changing the security settings or completely removing encryption from the document as long as the original document does not have a user password.

The user password protects the document, so that it only can be opened if the user or owner password is known. No PDF application program can open a user-password protected PDF document without providing the password. The security of such a document however strongly depends on the password itself. Like in most password related situations insecure passwords can easily be found programmatically. E.g. a brute force attempt testing all passwords which either exist as word in a dictionary or have less than six characters only takes minutes.

5.4 Fonts

Some features of the 3-Heights™ PDF Security API require fonts to be installed, e.g. when stamping text.

The location of the font directories depends on the operating system:

- Windows: `%SystemRoot%\Fonts` and directory `Fonts`, which must be a direct sub-directory of where the main DLL or executable resides.
- Mac: `/System/Library/Fonts` and `/Library/Fonts`
- Unix: `$PDFFONTDIR` or `/usr/lib/X11/fonts/Type1`

The fonts of the font directories and their properties are cached in a font cache, located in the files `font-database*` in the temporary files folder. The cache files have to be removed manually, if fonts are added or removed from these directories.

The directory for temporary files is determined as follows. The product checks for the existence of environment variables in the following order and uses the first path found:

- *Windows:*
  1. The path specified by the *TMP* environment variable.

2. The path specified by the *TEMP* environment variable.
3. The path specified by the *USERPROFILE* environment variable.
4. The Windows directory.

- *Unix:*
  1. The path specified by the *PDFTMPDIR* environment variable.
  2. The path specified by the *TMP* environment variable.
  3. The */tmp* directory.

Example: Command to remove the font cache files on Windows

```
del %TMP%\font-database*
```

5.5 Cryptographic Provider

In order to use the 3-Heights™ PDF Security API's cryptographic functions such as creating or validating digital signatures, a cryptographic provider is required. The cryptographic provider manages certificates, their private keys and implements cryptographic algorithms.

The 3-Heights™ PDF Security API can use various different cryptographic providers. The following list shows, for which type of signing certificate which provider can be used.

USB Token or Smart Card

These devices typically offer a PKCS#11 interface, which is the recommended way to use the certificate → PKCS#11 Provider.

On Windows, the certificate is usually also available in the Microsoft CryptoAPI. This provider is not recommended, unless you experience problems with your device's PKCS#11 interface.

If you need to sign documents on a non-Windows system with an USB token that does not come with middleware for your platform, you can use the 3-Heights™ Signature Creation and Validation Service.

If you need to sign documents on Windows in a non-interactive or locked session[1], use the 3-Heights™ Signature Creation and Validation Service.

Hardware Security Module (HSM)

HSMs always offer very good PKCS#11 support → PKCS#11 Provider

For more information and installation instructions see separate document TechNotePKCS11.pdf.

Soft Certificate

Soft certificates are typically PKCS#12 files that have the extension `.pfx` or `.p12` and contain the signing certificate as well as the private key and trust chain (issuer certificates). Soft certificate files cannot be used directly. Instead, they must be imported into the certificate store of a cryptographic provider.

- *All Platforms*: The recommended way of using soft certificates is to import them into a store that offers a PKCS#11 interface and use the PKCS#11 Provider. For example:
  - A HSM
  - openCryptoki on Linux
  - PKCS#11 softtoken on Solaris

  For more information and installation instructions of the above stores see separate document TechNotePKCS11.pdf.
- *Windows*: If no PKCS#11 provider is available, soft certificates can be imported into Windows certificate store, which can then be used as cryptographic provider → Microsoft CryptoAPI

Signature Service

Signature services are a convenient alternative to storing certificates and key material locally. The 3-Heights™ PDF Security API can use various different services whose configuration is explained in the following sections of this documentation:

- 3-Heights™ Signature Creation and Validation Service
- SwissSign Personal Signing Service
- SwissSign SuisseID Signing Service
- QuoVadis sealsign

[1] See the description of the 3-Heights™ Signature Creation and Validation Service for more details on this topic.

- Swisscom All-in Signing Service

Custom Signature Handler

If you want to create the cryptographic part of the signature yourself, i.e. you want to implement the cryptographic provider yourself, you can register a Custom Signature Handler. This is described in the respective subsection.

5.5.1 PKCS#11 Provider

PKCS#11 is a standard interface offered by most cryptographic devices such as HSMs, USB Tokens or sometimes even soft stores (e.g. openCryptoki).

More information on and installation instructions of the PKCS#11 provider of various cryptographic devices can be found in the separate document TechNotePKCS11.pdf.

Configuration

Provider Property Provider

The provider configuration string has the following syntax:

`"PathToDll;SlotId;Pin"`

- `PathToDll` is the path to driver library filename, which is provided by the manufacturer of the HSM, UBS token or smart card. Examples:
  - The SuisseID USB Tokens use *cvp11.dll*
  - The CardOS API from Atos (Siemens) uses *siecap11.dll*
  - The IBM 4758 cryptographic coprocessor uses *cryptoki.dll*
  - Devices from Aladdin Ltd. use *etpkcs11.dll*
- `SlotId` is optional, if it is not defined, it is searched for the first slot that contains a running token.
- `Pin` is optional, if it is not defined, the submission for the pin is activated via the pad of the token. If this is not supported by the token, the following error message is raised when signing: “Cannot access private key”.

Examples:

- `Provider = "\WINDOWS\system32\siecap11.dll;4;123456"`

Note: Some PKCS#11 drivers require the Terminate method to be called. Otherwise your application might crash upon termination.

The chapter Guidelines for Mass Signing contains important information to optimize performance when signing multiple documents.

Interoperability Support

The following cryptographic token interface (PKCS#11) products have been successfully tested:

- SafeNet Protect Server
- SafeNet Luna
- SafeNetAuthentication Client
- IBM OpenCrypTokI
- CryptoVision
- Siemens CardOS

Selecting a Certificate for Signing

The 3-Heights™ PDF Security API offers different ways to select a certificate. The product tries the first of the following selection strategies, for which the required values have been specified by the user.

1. **Certificate fingerprint**
   Property SignerFingerprint
   - SHA1 fingerprint of the certificate. The fingerprint is 20 bytes long and can be specified in hexadecimal string representation, e.g. “b5 e4 5c 98 5a 7e 05 ff f4 c6 a3 45 13 48 0b c6 9d e4 5d f5”. In Windows certificate store this is called “Thumbprint”, if “Thumbprint algorithm” is “sha1”.
2. **Certificate Issuer and SerialNumber**
   Properties Issuer and SerialNumber

- Certificate Issuer (e.g. “QV Schweiz CA”), in Windows certificate store this is called “Issued By”.
- Serial number of the certificate (hexadecimal string representation, e.g. “4c 05 58 fb”). This is a unique number assigned to the certificate by its issuer. In Windows certificate store this is the field called “Serial number” in the certificate’s “Details” tab.

3. **Certificate Name and optionally Issuer**
   Properties Name and Issuer
   - Common Name of the certificate (e.g. “PDF Tools AG”), in Windows certificate store this is called “Issued To”.
   - Optional: Certificate Issuer (e.g. “QV Schweiz CA”), in Windows certificate store this is called “Issued By”.

Using PKCS#11 stores with missing issuer certificates

Some PKCS#11 devices contain the signing certificate only. However, in order to embed revocation information it is important, that the issuer certificates, i.e. the whole trust chain, is available as well.

On Windows, missing issuer certificates can be loaded from the Windows certificate store. So the missing certificates can be installed as follows:

1. Get the certificates of the trust chain. You can download them from the website of your certificate provider or do the following:
   (a) Sign a document and open the output in Adobe Acrobat
   (b) Go to “Signature Properties” and then view the signer’s certificate
   (c) Select a certificate of the trust chain
   (d) Export the certificate as “Certificate File” (extension .cer)
   (e) Do this for all certificates of the trust chain
2. Open the exported files by double clicking on them in the Windows Explorer
3. Click button “Install Certificate...”
4. Select “automatically select the certificate store based on the type of certificate” and finish import

5.5.2 Microsoft CryptoAPI Provider

Microsoft CryptoAPI (MS-CAPI, CAPI) offers access to the certificates stored in the Windows certificate store and other devices, such as USB tokens, with Windows integration. Microsoft CryptoAPI does not support some new cryptographic algorithms. Therefore it is recommended to use the PKCS#11 Provider if possible.

Configuration

Provider Property Provider

The provider configuration string has the following syntax:

```
"[ProviderType:]Provider[;PIN]"
```

The ProviderType and PIN are optional. The corresponding drivers must be installed on Windows.
Examples:

- `Provider = "Microsoft Base Cryptographic Provider v1.0"`
- `Provider = "Microsoft Strong Cryptographic Provider"`
- `Provider = "PROV_RSA_AES:Microsoft Enhanced RSA and AES Cryptographic Provider"`

The provider type PROV_RSA_AES supports the SHA-2 hash algorithms for signature validation (not signature creation). This provider type is recommended in order to validate signatures if no PKCS#11 device is available.
Optionally, when using an advanced certificate, the pin code can be passed as an additional, semi-column separated parameter. This does not work with qualified certificates, because they always require the pin code to be entered manually and every time. If the name of the provider is omitted, the default provider is used.
Examples, “123456” being the pin code:

- `Provider = "Microsoft Base Cryptographic Provider v1.0;123456"`
- `Provider = ";123456"`

Certificate Store Property Store

The value for the certificate store depends on the OS. Supported values are: “CA”, “MY” and “ROOT”. For signature creation the default store “MY” is usually the right choice.

Store Location Property StoreLocation

Either of the following store locations

- Local Machine
- Current User (default)

Usually personal certificates are stored in the current user location and company-wide certificates are stored under local machine.

The current user's store is only available, if the user profile has been loaded. This may not be the case in certain environments such as within an IIS web application or COM+ applications. Use the store of the Local Machine, if the user profile cannot be loaded.

Certificates in the store Local Machine are available to all users. However, in order to sign a document, you need access to the signing certificate's private key. The private key is protected by Windows ACLs and typically readable for Administrators only. Use the Microsoft Management Console (mmc.exe) in order to grant access to the private key for other users as follows: Add the Certificates Snap-in for the certificates on Local Machine. Right-click on the signing certificate, click on "All Tasks" and then "Manage Private Keys..." where you can set the permissions.

Selecting a Certificate for Signing

First, the certificate store defined by the provider is used. Within the store the selection of the signing certificate works the same as with the PKCS#11 provider, which is described here: Selecting a Certificate for Signing

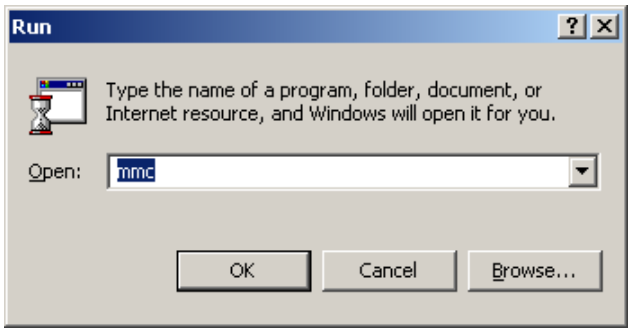
Certificates

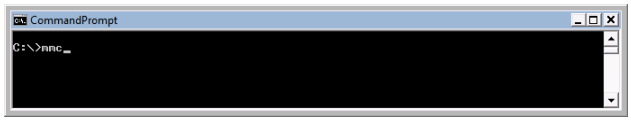
In order to sign a PDF document, a valid, existing certificate name must be provided and its private key must be available.

There are various ways to create or obtain a certificate. How this is done is not described in this document. This document describes the requirements for, and how to use the certificate.

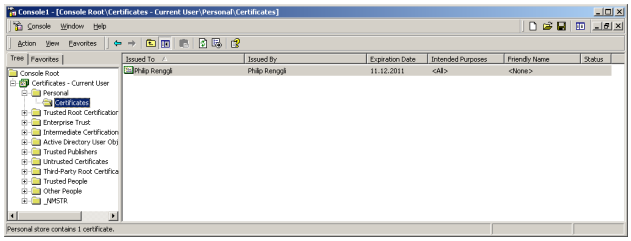
On the Windows operating system certificates can be listed by the Microsoft Management Console (MMC), which is provided by Windows. In order to see the certificates available on the system, do the following steps:

1. To launch the MMC, go to Start ->Run...->type "mmc", or start a Command Prompt and type "mmc".

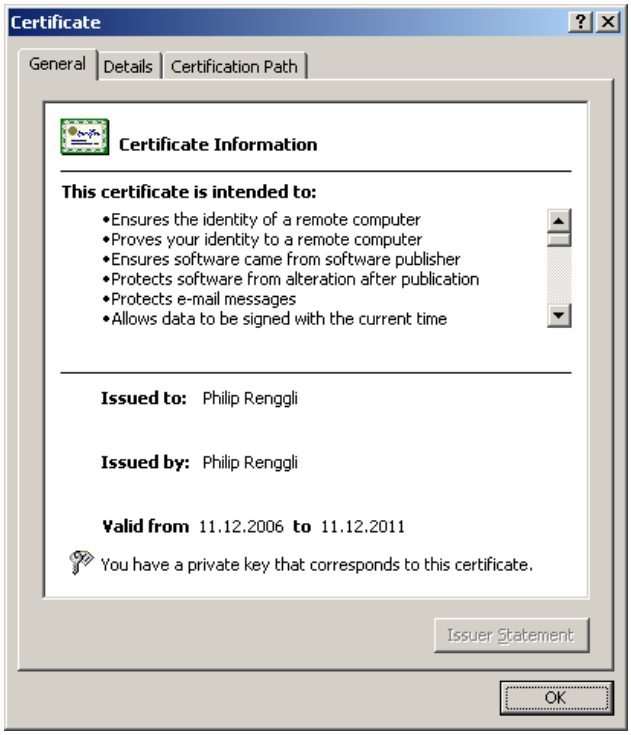




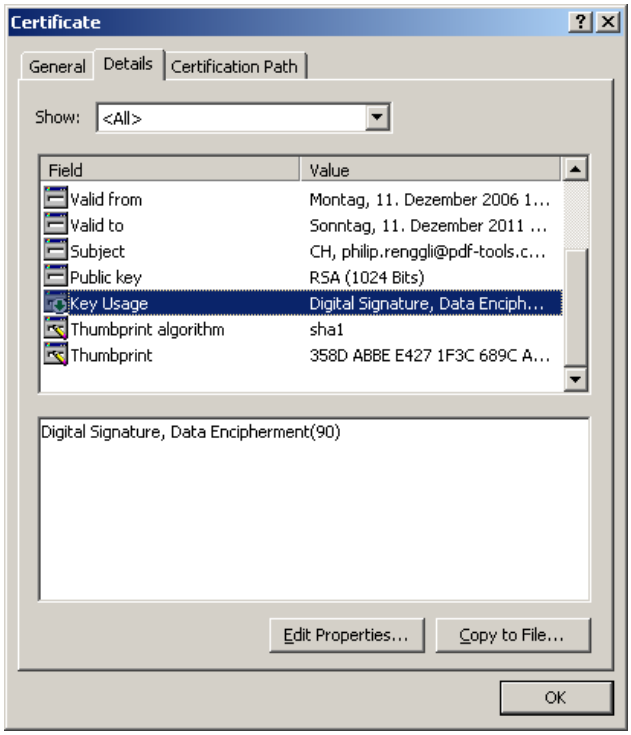
2. Under "File" ->"Add/Remove Snap-in"
3. Choose "Certificates" and click the "Add" button
4. In the next window choose to manage certificates for "My user account"
5. Click "Finish"
6. The certificate must be listed under the root "Certificates - Current User", for example as shown in the screenshot below:



7. Double-click the certificate to open. The certificate name corresponds to the value “Issued to:”.

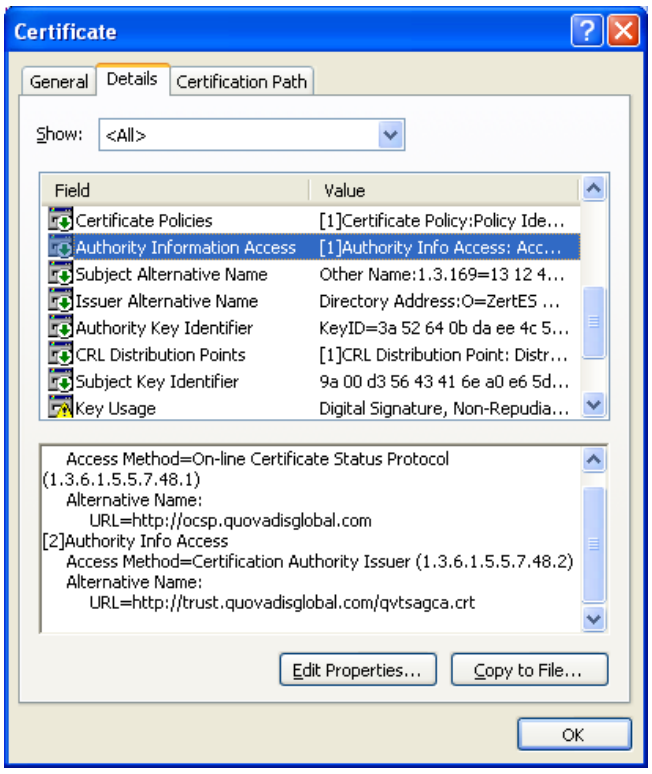

Certificate
General
Details
Certification Path
Certificate Information
This certificate is intended to:
•Ensures the identity of a remote computer
•Proves your identity to a remote computer
•Ensures software came from software publisher
•Protects software from alteration after publication
•Protects e-mail messages
•Allows data to be signed with the current time
Issued to: Philip Renggli
Issued by: Philip Renggli
Valid from 11.12.2006 to 11.12.2011
You have a private key that corresponds to this certificate.
Issuer Statement
OK

8. In the tab Detail of the certificate, there is a field named “Key Usage”. This field must contain the value “Digital Signature”. Additional values are optional, see also screenshot. You must have the private key that corresponds to this certificate.



Qualified Certificates

A qualified certificate can be obtained from a certificate authority (CA). Besides the requirements listed in the previous chapter it has the additional requirement to contain the key “Authority Information Access” which contains the information about the OCSP server.



5.5.3 3-Heights™ Signature Creation and Validation Service

The 3-Heights™ Signature Creation and Validation Service provides HTTP protocol based remote access to cryptographic providers such as smartcards, USB tokens, and other cryptographic infrastructure such as HSMs.

Use of the 3-Heights™ Signature Creation and Validation Service provides the following advantages:

1. By means of this service the tokens can be hosted centrally and used by any client computer which has access to the service.
2. Cryptographic devices that can be used on Windows only can be made accessible to siging processes running on Non-Windows systems.
3. Cryptographic devices can be made accessible to processes running in non-interactive sessions. Many cryptographic devices must always be used in an interactive session for two reasons. First, the middleware requires the user to enter the pin interactively to create a qualified electronic signature. Second, USB tokens and smart cards are managed by Windows such that the device is available only to the user currently using the computer’s console. Therefore, services, remotely logged in users and applications running in locked sessions have no access to the device.

Note that this is a separate product and this chapter describes its usage with the 3-Heights™ PDF Security API only. For more information on the 3-Heights™ Signature Creation and Validation Service and installation instructions, please refer to its separate user manual.

Configuration

Provider Property Provider

The provider configuration string has the following syntax:

`"http://server.mydomain.com:<port>/<token>;<password>"`

Where:

- `server.mydomain.com` is the hostname of the server
- `<port>` is optional, port of the server.
- `<token>` the ID of the token.

- `<password>` password of the token.

Examples:

- `Provider = "http://server.mydomain.com:8080/0001;pass01"`

A more detailed description can be found in the user manual of the 3-Heights™ Signature Creation and Validation Service.

Selecting a Certificate for Signing

Selection of the signing certificate works the same as if the token was used directly: Selecting a Certificate for Signing.

5.5.4 SwissSign Personal Signing Service

Provider Property Provider

The provider configuration string contains the URL to the service endpoint.

Provider Configuration

The provider can be configured using provider session properties.

There are two types of properties:

- *String Properties:*
  String properties are set using method SetSessionPropertyString.
- *File Properties:*
  File properties are set using method SetSessionPropertyString with a file name parameter. Alternatively the file can be passed in-memory as byte array using the method SetSessionPropertyBytes.

Name	Type	Required	Value
Identity	String	required	The identity of your signing certificate. Example: `My Company:Signing Cert 1`
DSSProfile	String	required	`http://dss.swisssign.net/dss/profile/pades/1.0`
SSLClientCertificate	File	required	SSL client certificate in PKCS#12 Format (.p12, .pfx). File must contain the certificate itself, all certificates of the trust chain and the private key.
SSLClientCertificatePassword	String	optional	Password to decrypt the private key of the SLL client certificate.
SSLServerCertificate	File	recommended	Certificate of the server or its issuer (CA) certificate in DER Format (.der, .cer) Note: If this property is not set, the server certificate is not verified at all!
RequestID	String	recommended	Any string that can be used to track the request. Example: An UUID like `AE57F021-C0EB-4AE0-8E5E-67FB93E5BC7F`

Signature Configuration

The signature can be customized using standard properties of the 3-Heights™ PDF Security API.

Description	Required	Value	Setting
Common Name	required	The name of the signer should be set[2].	Property Name.
Time-stamp	optional	`"urn:ietf:rfc:3161"`	Property TimeStampURL
Revocation Info	optional	`true` to embed OCSP responses or CRL.	Property EmbedRevocationInfo
Visual Appearance	optional	See separate chapter on creating a visual appearance.	

Proxy Configuration

If a proxy is used for the connection to the service, see chapter How to Use a Proxy for more information.

[2]This parameter is not used for certificate selection, but for the signature appearance and signature description in the PDF only.

5.5.5 SwissSign SuisseID Signing Service

In order to use the SuisseID Signing Service, please contact Swiss Post Solutions AG (suisseid@post.ch) to obtain access credentials. Prior to invoking the SuisseID Signing Service, user authentication via the SuisseID Identity Provider (IDP) is a pre-requisite. So the calling application must integrate via SAML (e.g. SuisseID SDK) with the SuisseID Identity Provider. The IDP issues SAML tokens upon successful user authentication.

Note that the name of the signature should be the signer's name (e.g. "<givenname> <surname>"). The signer's name can be retrieved for the SAML token as the IDP provides this as qualified attributes (yellowid verified).

Provider Property Provider

The provider configuration string contains the URL to the service Endpoint.

Provider Configuration

The provider can be configured using provider session properties.

There are two types of properties:

- *String Properties:*
  String properties are set using method SetSessionPropertyString.
- *File Properties:*
  File properties are set using method SetSessionPropertyString with a file name parameter. Alternatively the file can be passed in-memory as byte array using the method SetSessionPropertyBytes.

Name	Type	Required	Value
SAMLToken	File	required	SAML token issued by the SuisseID Identity Provider (IDP). Example: `C:\temp\my-saml.xml` Note: The SAML token received from the IDP is a signed XML and must not be modified in any way. For example, the token should not be read into a string or XML object and re-serialized.
SSLClientCertificate	File	required	SSL client certificate in PKCS#12 Format (.p12, .pfx). File must contain the certificate itself, all certificates of the trust chain and the private key.
SSLClientCertificatePassword	String	optional	Password to decrypt the private key of the SLL client certificate.
SSLServerCertificate	File	recommended	Certificate of the server or its issuer (CA) certificate in DER Format (.der, .cer) Note: If this property is not set, the server certificate is not verified at all!

Signature Configuration

The signature can be customized using standard properties.

Description	Required	Value	Setting
Common Name	required	The name of the signer should be set[3].	Property Name.
Time-stamp	optional	`"urn:ietf:rfc:3161"`	Property TimeStampURL
Revocation Info	optional	`true` to embed OCSP responses or CRL.	Property EmbedRevocationInfo
Visual Appearance	optional	See separate chapter on creating a visual appearance.	

[3]This parameter is not used for certificate selection, but for the signature appearance and signature description in the PDF only.

Proxy Configuration

If a proxy is used for the connection to the service, see chapter How to Use a Proxy for more information.

5.5.6 QuoVadis sealsign

Provider Property Provider

The provider configuration string contains the URL to the QuoVadis sealsign service. For example:
`"http://services.sealsignportal.com:18080/sealsign/ws/BrokerClient"`

Provider Configuration

The provider can be configured using provider session properties that can be set using the method SetSessionPropertyString.

Name	Type	Required	Value
Identity	String	required	The account ID is the unique name of the account specified on the server. Example: `Rigora`
Profile	String	required	The profile identifies the signature specifications by a unique name. Example: `Default`
secret	String	required	The secret is the password which secures the access to the account. Example: `NeE=EKEd33FeCk70`
clientId	String	optional	A client ID can be used to help separating access and creating better statistics. If specified in the account configuration it is necessary to provide this value. Example: `3949-4929-3179-2818`
pin	String	required	The PIN code is required to activate the signing key. Example: `123456`
MessageDigestAlgorithm	String	optional	The message digest algorithm to use. Note that the supported algorithms depend on the provider. Default: `SHA-256` Alternatives: `SHA-1`, `SHA-384`, `SHA-512`, `RIPEMD-160`, `RIPEMD-256`

Signature Configuration

The signature can be customized using standard properties.

Description	Required	Value	Setting
Common Name	required	The name of the signer should be set[4].	Property Name.
Time-stamp	-	Not available.	
Revocation Info	optional	`true` to embed OCSP responses or CRL.	Property EmbedRevocationInfo
Visual Appearance	optional	See separate chapter on creating a visual appearance.	

Proxy Configuration

If a proxy is used for the connection to the service, see chapter How to Use a Proxy for more information.

[4]This parameter is not used for certificate selection, but for the signature appearance and signature description in the PDF only.

5.5.7 Swisscom All-in Signing Service

General Properties

To use the signature service, the following general properties have to be set:

Description	Required	Value	Setting
Common Name	required	Name of the signer[5].	Property Name
Provider	required	The service endpoint URL of the REST service. Example: `https://ais.swisscom.com/AIS-Server/rs/v1.0/sign`	Property Provider
Time-stamp	optional	`"urn:ietf:rfc:3161"`	Property TimeStampURL
Revocation Info	optional	`true` to embed OCSP responses	Property EmbedRevocationInfo

If a proxy is used for the connection to the service, see chapter How to Use a Proxy for more information.

Provider Session Properties

In addition to the general properties, a few provider specific session properties have to be set.

There are two types of properties:

- *String Properties:*
  String properties are set using method SetSessionPropertyString.
- *File Properties:*
  File properties are set using method SetSessionPropertyString with a file name parameter. Alternatively the file can be passed in-memory as byte array using the method SetSessionPropertyBytes.

Name	Type	Required	Value
DSSProfile	String	required	`http://ais.swisscom.ch/1.0`
SSLClientCertificate	File	required	SSL client certificate in PKCS#12 Format (.p12, .pfx). File must contain the certificate itself, all certificates of the trust chain and the private key.
SSLClientCertificatePassword	String	optional	Password to decrypt the private key of the SLL client certificate.
SSLServerCertificate	File	recommended	Certificate of the server or its issuer (CA) certificate in DER Format (.der, .cer) Note: If this property is not set, the server certificate is not verified at all!
Identity	String	required	The identity string as provided by Swisscom: `<customer name>:<key identity>`
RequestID	String	recommended	Any string that can be used to track the request. Example: An UUID like `AE57F021-C0EB-4AE0-8E5E-67FB93E5BC7F`

[5]This parameter is not used for certificate selection, but for the signature appearance and signature description in the PDF only.

On-Demand Certificates

To request an on-demand certificate, the following additional property has to be set:

Name	Type	Required	Value
SwisscomAllInOnDemandDN	String	required	The requested distinguished name. Example: `cn=Hans Muster,o=ACME,c=CH`

Step-Up Authorization using Mobile-ID

To use the step-up authorization, the following additional properties have to be set:

Name	Type	Required	Value
SwisscomAllInMSISDN	String	required	Mobile phone number. Example: `+41798765432`
SwisscomAllInMessage	String	required	The message to be displayed on the mobile phone. Example: `Pipapo halolu.`
SwisscomAllInLanguage	String	required	The language of the message. Example: `DE`

Those properties have to comply with the Swisscom Mobile-ID specification.

5.5.8 Custom Signature Handler

The 3-Heights™ PDF Security API provides the capability of replacing the default built-in signature handler with a custom signature handler. A custom signature handler has full control over the creation and validation of the cryptographic part of a signature. This makes it possible to implement proprietary signing algorithms.

The custom signature handler must implement a C interface as described in the header file `pdfsignaturehandler.h`. It can be registered using a call to `PdfRegisterSignatureHandler()` during the initialization of the 3-Heights™ PDF Security API. When using a custom signature handler, it is important that this call be made before using the API for signing.

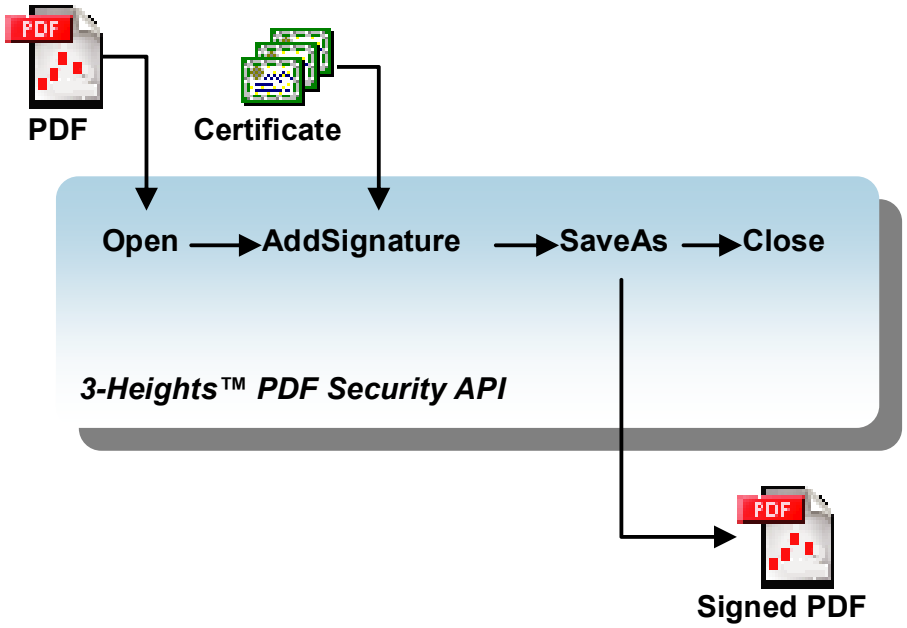
This allows for treating the PDF and signature technologies separately and also provides an easy way to replace a signature handler.

5.6 How to Create Digital Signatures

This chapter describes the steps that are required to create different types of digital signatures. A good introductory example can be found in the chapter Digital Signatures.

5.6.1 How to Sign a PDF Document

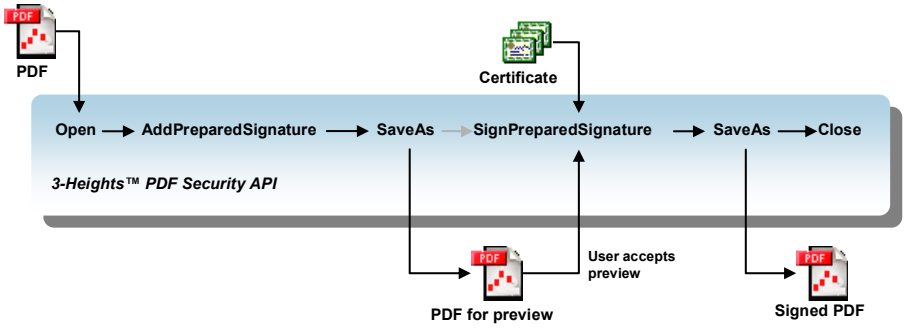
As we saw in the chapter Digital Signatures, the process steps to add a signature are as shown in the graphic below:



1. A PDF input-document is opened
2. A signature is created and added using a certificate
3. A new, signed PDF output-document is created
4. The input-document is closed

5.6.2 How to create a Preview of a signed Document

The 3-Heights™ PDF Security API provides the possibility to create a PDF document with a visual appearance of a digital signature without actually signing the document. This document can be used for a preview. If the preview is accepted, the document can be signed without visually change the document. The process steps to prepare a document for signing and actually sign it upon approval of the user are as shown in the graphic below:



1. A PDF input-document is opened
2. A digital signature is prepared and a visual appearance is generated
3. A new preview-PDF output-document is created, this document does not contain a digital signature, however it contains a placeholder for a signature
4. If the preview-PDF is approved, the document is signed using a certificate
5. A new, signed PDF output-document is created, which looks identical to the preview-PDF
6. The input-document is closed

5.6.3 How to Create a PAdES LTV Signature

In order to create a PAdES LTV signature, the following is required:

1. *An advanced or qualified signing certificate.*
   For requirements and preparation steps see the sample in chapter Digital Signatures. Make sure the store of your cryptographic provider contains all certificates of the trust chain, including the root certificate.
2. *Embed revocation information.*
   Do not set the property EmbedRevocationInfo to false.
3. *Add a Time-stamp.*
   Use the property TimeStampURL.
4. *Proper error handling.*
   A proper error handling is crucial in order to ensure the creation of correctly signed documents. The output document was signed successfully, if and only if the method SaveAs returns true and the property ErrorCode does not have any of the following values: SIG_CREA_E_OCSP, SIG_CREA_E_CRL, or SIG_CREA_E_TSP.

5.6.4 How to Create a Time-stamp Signature

For a Time-stamp signature no local signing certificate is required. Instead the Time-stamp signature requested from the Time-stamp Authority (TSA) is embedded into the document.

Example: Create a Time-stamp signature using the method AddTimeStampSignature.

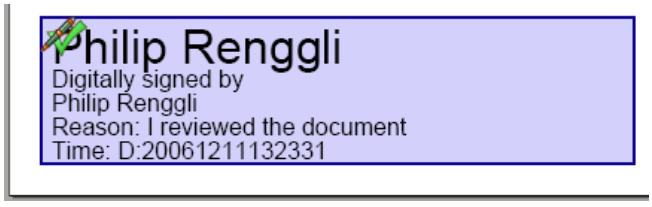
```
Dim Document As New PDFSECUREAPILib.PdfSecure
Document.Open "input.pdf", ""

Dim Signature As New PDFSECUREAPILib.PdfSignature
Signature.Provider = "cvp11.dll"
Signature.TimeStampURL = "http://server.mydomain.com:80/tsa"

Document.AddTimeStampSignature Signature
Document.SaveAs "output.pdf"
Document.Close
```

A Cryptographic Provider is required on non-Windows systems only.

5.6.5 How to Create a Visual Appearance of a Signature

Each signature may have a visual appearance on a page of the document. The visual appearance is optional and has no effect on the validity of the signature. Because of this and because a visual appearance may cover important content of the page, many applications choose to create an invisible signature.

By default, the 3-Heights™ PDF Security API creates an appearance in the lower left corner of the last page, which looks as shown below:



How to Create an Invisible Signature

Invisible signatures have no visual appearance and can be created by setting an empty rectangle using the Property Rect:

```
signature.Rect = New PDFRect(0, 0, 0, 0)
```

How to Create a Visual Appearance

Different properties of the visual appearance can be specified.

Page and Position
: See properties PageNo and Rect.

Color
: See properties FillColor and StrokeColor.

Line Width
: The line width of the background rectangle, see property LineWidth.

Text
: Two text fragments can be set using two different fonts and font sizes, see properties Text1, Text2, FontName1, FontName2, FontSize1 and FontSize2.

Background image
: See property ImageFileName.

5.6.6 Guidelines for Mass Signing

This section provides some guidelines for mass signing using the 3-Heights™ PDF Security API.

Keep the session to the security device open for multiple sign operations

Creating and ending the session to the security device is a complex operation. By re-using the session for multiple sign operations, performance can be improved:

1. Create a `PdfSecure` object
2. Open the session to the provider using `BeginSession`
3. Use the PdfSecureobject to sign multiple documents
4. Close the session to the provider using `EndSession`
5. Dispose of the `PdfSecure` object

Signing concurrently using multiple threads

The 3-Heights™ PDF Security API is thread-safe. Each `PdfSecure` object should be used in one thread at the time only. It is recommended that each thread has a separate `PdfSecure` object.

The performance improvement when signing concurrently using multiple threads depends mainly on the security device used. Typically the improvement is large for HSMs and small for USB Tokens.

Thread safety with a PKCS#11 provider

The PKCS#11 standard specifies, that "an application can specify that it will be accessing the library concurrently from multiple threads, and the library must [...] ensure proper thread-safe behavior." However, some PKCS#11 provider (middleware) implementations are not thread-safe. For this reason, the 3-Heights™ PDF Security API synchronizes all access to the same provider (middleware and slot id).

If your middleware is thread-safe, you can enable full parallel usage of the cryptographic device by setting the session property `"LOCKING_OK"` to the value `"true"` using the method `SetSessionPropertyString`:

Example: Enable parallel access to the cryptographic device.

```
doc.SetSessionPropertyString("LOCKING_OK", "true");
```

5.6.7 Miscellaneous

Caching of CRLs, OCSP and TSP Reponses

In order to improve the speed when mass signing, the 3-Heights™ PDF Security API provides a caching algorithm to store CRL (Certificate Revocation List), OCSP (Online Certificate Status Protocol), TSP (Time-stamp Protocol) and data from signature services. This data is usually valid over period of time that is defined by the protocol, which is normally at least 24 hours. Caching improves the speed, because there are situations when the server does not need to be contacted for every digital signature. The following caches are stored automatically by the 3-Heights™ PDF Security API at the indicated locations within the directory for temporary files:

OCSP responses: *ocsp/server-hash.der*

CRL: *crl/server.der*

TSP responses[6] *tsp/server.der*

Service data: *sig/hash.bin*

The caches can be cleared by deleting the files. Usage of the caches can be deactivated by setting the NoCache flag. The files are updated if the current date and time exceeds the "next update" field in the OCSP or CRL response respectively or the cached data was downloaded more than 24 hours ago.

The directory for temporary files is determined as follows. The product checks for the existence of environment variables in the following order and uses the first path found:

- *Windows:*
  1. The path specified by the *TMP* environment variable.
  2. The path specified by the *TEMP* environment variable.
  3. The path specified by the *USERPROFILE* environment variable.
  4. The Windows directory.
- *Unix:*
  1. The path specified by the *PDFTMPDIR* environment variable.
  2. The path specified by the *TMP* environment variable.
  3. The */tmp* directory.

How to Use a Proxy

The 3-Heights™ PDF Security API can use a proxy server for all communication to remote servers, e.g. to download CRL or for communication to a signature service. The proxy server can be configured using the provider session property *Proxy*. The property's value must be a string with the following syntax:

`http[s]://<user:password>@host:<port>`

Where:

- `http` / `https`: Protocol for connection to proxy.
- `user:password` (optional): Credentials for connection to proxy (basic authorization).
- `host`: Hostname of proxy.
- `port`: Port for connection to proxy.

For SSL connections, e.g. to a signature service, the proxy must allow the HTTP CONNECT request to the signature service.

Example: Configuration of a proxy server that is called `myproxy` and accepts HTTP connections on port `8080`.

```
doc.SetSessionPropertyString "Proxy" "http://myproxy:8080"
```

[6] The sizes of the TSP responses are cached only. Cached TSP responses cannot be embedded but used for the computation of the signature length only.

Configuration of Proxy Server and Firewall

For the application of a Time-stamp or online verification of certificates, the signature software requires access to the server of the certificates' issuer (e. g. http://ocsp.quovadisglobal.com or http://platinum-qualified-g2.ocsp.swisssign.net/) via HTTP. The URL for verification is stored in the certificate; the URL for Time-stamp services is provided by the issuer. In case these functions are not configured, no access is required.

In organizations where a web proxy is in used, it must be ensured that the required MIME types are supported. These are:

OCSP

- `application/ocsp-request`
- `application/ocsp-response`

Time-stamp

- `application/timestamp-query`
- `application/timestamp-reply`

Signature services

- Signature service specific MIME types.

5.7 How to Validate Digital Signatures

Validation of a Qualified Electronic Signature

There are basically three items that need to be validated:

1. Trust Chain
2. Revocation Information (optional)
3. Time-stamp (optional)

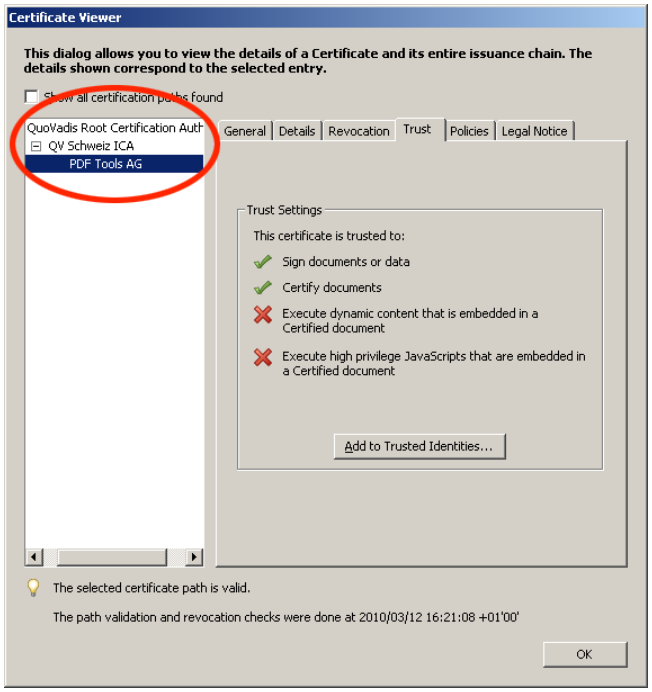
Validation can be in different ways, e.g. Adobe Acrobat, from which the screenshots below are taken.

Trust Chain

Before the trust chain can be validated, ensure the root certificate is trusted. There are different ways to add a certificate as trusted root certificate. The best way on Windows is this:

1. Retrieve a copy of the certificate containing a public key. This can be done be requesting it from the issuer (your CA) or by exporting it from an existing signature to a file (*CertExchange.cer*). Ensure you are not installing a malicious certificate!
2. Add the certificate to the trusted root certificates. If you have the certificate available as file, you can simply double-click it to install it.

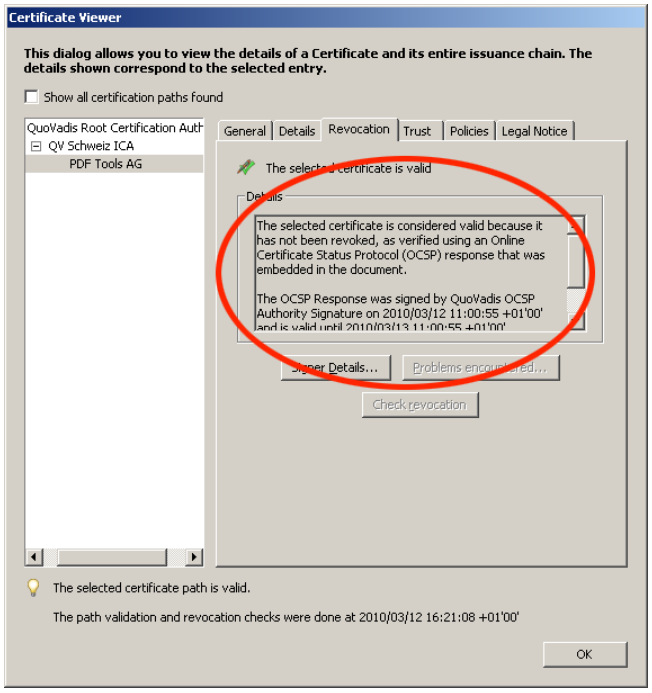
After that you can validate the signature, e.g. by open the PDF document in Adobe Acrobat, right-click the signature and select “Validate”, then select “Properties” and select the tab “Trust”. There the certificate should be trusted to “sign documents or data”.


Certificate Viewer
This dialog allows you to view the details of a Certificate and its entire issuance chain. The details shown correspond to the selected entry.
Show all certification paths found
QuoVadis Root Certification Auth
QV Schweiz ICA
PDF Tools AG
General
Details
Revocation
Trust
Policies
Legal Notice
Trust Settings
This certificate is trusted to:
Sign documents or data
Certify documents
Execute dynamic content that is embedded in a Certified document
Execute high privilege JavaScripts that are embedded in a Certified document
Add to Trusted Identities...
The selected certificate path is valid.
The path validation and revocation checks were done at 2010/03/12 16:21:08 +01'00'
OK

Revocation Information

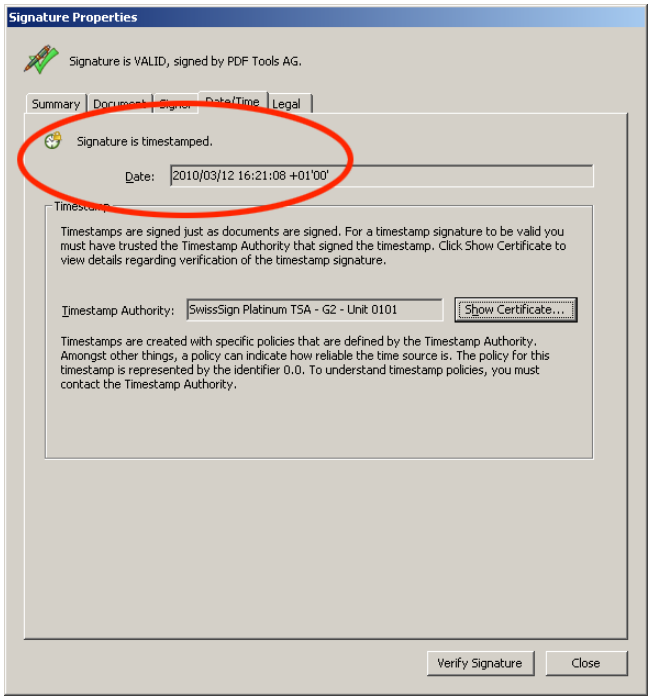
An OCSP response or CRL must be available. This is shown in the tab “Revocation”. The details should mention that “the certificate is considered *valid*”.

The presence of revocation information must be checked for the signing certificate and all certificates of its trust chain except for the root certificate.


Certificate Viewer
This dialog allows you to view the details of a Certificate and its entire issuance chain. The details shown correspond to the selected entry.
Show all certification paths found
QuoVadis Root Certification Auth
QV Schweiz ICA
PDF Tools AG
General
Details
Revocation
Trust
Policies
Legal Notice
The selected certificate is valid
Details
The selected certificate is considered valid because it has not been revoked, as verified using an Online Certificate Status Protocol (OCSP) response that was embedded in the document.
The OCSP Response was signed by QuoVadis OCSP Authority Signature on 2010/03/12 11:00:55 +01'00'
Signer Details...
Problems encountered...
Check revocation
The selected certificate path is valid.
The path validation and revocation checks were done at 2010/03/12 16:21:08 +01'00'
OK

Time-stamp

The signature can optionally contain a Time-stamp. This is shown in the tab “Date/Time”. The certificate of the Time-stamp server must also be trusted, i.e. its trust chain should be validated as described in the section Trust Chain above.


Signature Properties
Signature is VALID, signed by PDF Tools AG.
Summary
Date/Time
Legal
Signature is timestamped.
Date: 2010/03/12 16:21:08 +01'00'
Timestamps are signed just as documents are signed. For a timestamp signature to be valid you must have trusted the Timestamp Authority that signed the timestamp. Click Show Certificate to view details regarding verification of the timestamp signature.
Timestamp Authority: SwissSign Platinum TSA - G2 - Unit 0101
Show Certificate...
Timestamps are created with specific policies that are defined by the Timestamp Authority. Amongst other things, a policy can indicate how reliable the time source is. The policy for this timestamp is represented by the identifier 0.0. To understand timestamp policies, you must contact the Timestamp Authority.
Verify Signature
Close

Validation of a PAdES LTV Signature

Verifying if a signature conforms to the PAdES LTV standard is similar to validating a Qualified Electronic Signature.

The following must be checked:

1. Trust Chain
2. Revocation information
3. Time-stamp
4. LTV expiration date
5. Other PAdES Requirements

Trust Chain

Trust chain validation works the same as for validating Qualified Electronic Signatures.

Revocation information

Revocation information (OCPS response or CRL) must be valid and embedded into the signature. In the details, verify that the revocation check was performed using data that was "was *embedded in the signature*". Revocation information that "was contained in the *local cache*" or "was requested *online*" is not embedded into the signature and does not meet PAdES LTV requirements.

Time-stamp

A Time-stamp must be embedded and validated as described for validating Qualified Electronic Signatures. If a document contains multiple Time-stamps, all but the latest one must contain revocation information.

LTV expiration date

The long term validation ability expires with the expiration of the signing certificate of the latest Time-stamp. The life-time of the protection can be further extended beyond the life-of the last Time-stamp applied by adding further DSS information to validate the previous last Time-stamp along with a new Time-stamp.

Other PAdES Requirements

Certain other PAdES requirements, such as requirements on the PKCS#7 CMS, cannot be validated using Adobe Acrobat. For this, use the 3-Heights™ PDF Security API for validation. See method ValidateSignature of the PdfSecure interface.

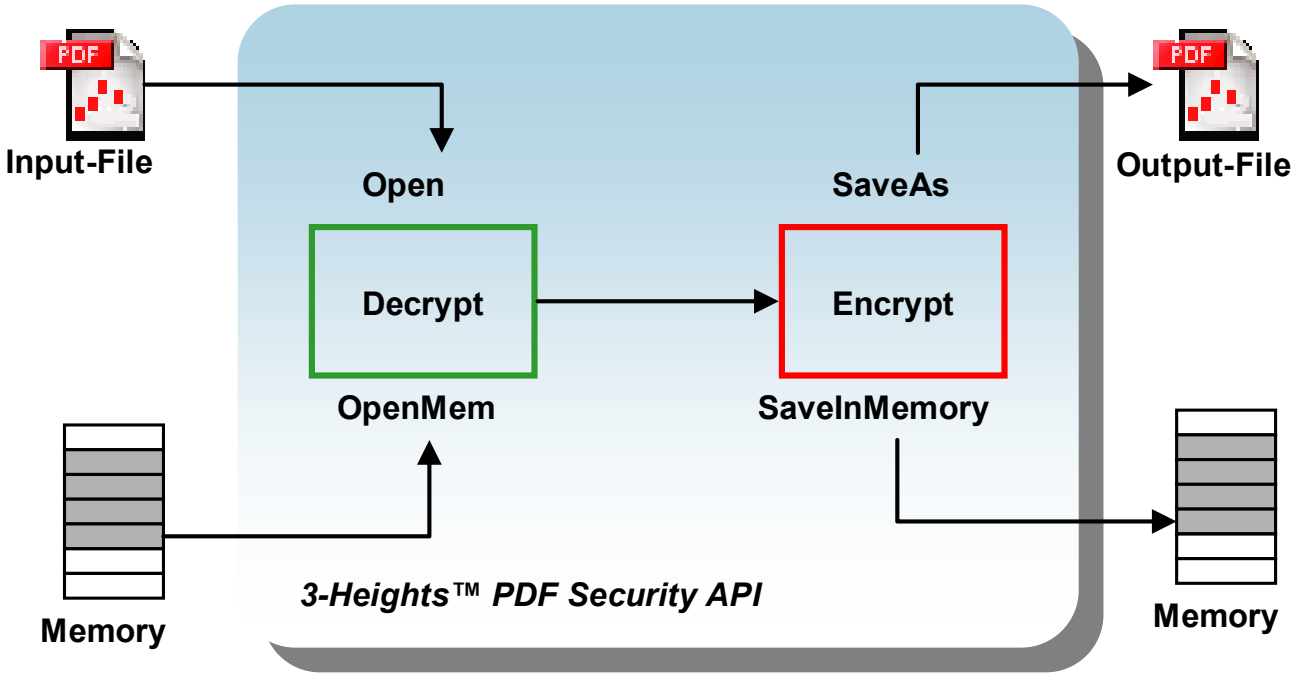
5.8 Advanced Guide

How to Use the in-Memory Functions

The 3-Heights™ PDF Security API always requires two PDF documents. A PDF input-document from which it reads and a PDF output-document to where the result is saved to.

To open from and save to files, the functions Open and SaveAs are used. These two functions are described in the chapters How to read an encrypted PDF and How to encrypt a PDF.

Instead of accessing files, the documents can be read from and written to in-memory. The corresponding functions are OpenMem and SaveInMemory.



Once the output-document is saved to memory using SaveInMemory, that memory block can be accessed using the function GetPDF.

A call sequence to create a first PDFSecure object that opens a PDF from file and stores its output inmemory and then a second object, which reads that in-memory document and saves it back to a file looks like this:

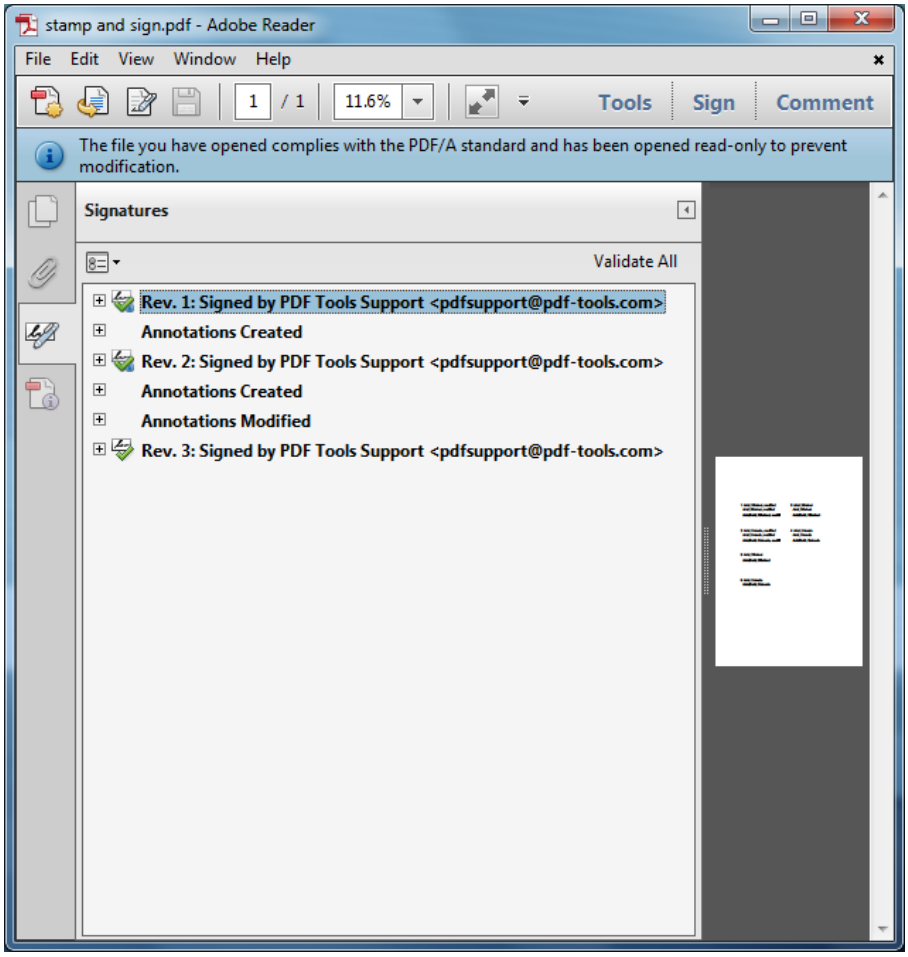
```
PDFSecure1.Open(InputFile)
PDFSecure1.SaveInMemory()
PDFSecure1.Close()
PDFSecure2.OpenMem(PDFSecure1.GetPDF())
PDFSecure2.SaveAs(OutputFile)
PDFSecure2.Close()
```

This call sequence of course does not make much sense. It's merely used to illustrate how to use of the in-memory functions. In a real application, the in-memory document is read from another application or a database.

5.9 Stamping

The 3-Heights™ PDF Security API can add new content such as text or images to the output document. This process is called stamping. The content of previously applied stamps can be modified.

The 3-Heights™ PDF Security API can sign and stamp documents in one step. In order to not invalidate existing signatures, stamps can be modified and created using stamp annotations with an incremental update to the input document. An example of this can be seen in the screenshot below.



Stamp File Syntax

Stamps are described with xml data that is passed to the 3-Heights™ PDF Security API either as file using the method `AddStamps` or as memory block using the method `AddStampsMem`.

A stamp file can contain one or more stamps. Each stamp is defined by a `ps:stamp` tag that specifies the stamp's size, position, and pages to which it is applied to.

Each stamp contains a number of content operators that define the appearance (i.e. the content) of the stamp. The content operators are applied in the order they appear within `ps:stamp` where each content element is drawn over all previous elements (i.e. increasing z-order).

Tags, Attributes-Names, Attribute-Values

`ps:pdfstamp`	The *Root Tag* for the PDF stamps. The tag may contain multiple stamps.
`xmln:ps http://www.pdf-tools.com/pdfstamp/`	
`ps:stamp`	*Stamp*: A stamp contains a collection of content operators.
`page first, even, odd`	The pages to which the stamp is to be applied. Comma-separated combinations are allowed. first First page last Last page odd Only odd pages including first page and last page in case it is odd even Only even pages including last page in case it is even all All pages not_first First page excluded not_last Last page excluded
`name name`	Identifier of the stamp optional, must be less than 127 characters, must be unique, see note 2 below.
`relativepos x y`	Relative position x and y of the stamp with regards to the page. Positive values of x and y define the distances of the stamp to the left and lower, negative values to the right and upper page boundary respectively. The positioning algorithm works best for stamp rotation angles that are a multiple of 90 (see rotate attribute). x or y is ignored, if respective align is used. Example 1: `relativepos=" 10 -10"` places stamp in upper left corner of page Example 2: `relativepos="-10 -10"` places stamp in upper right corner of page Example 3: `relativepos=" 10 10"` places stamp in lower left corner of page Example 4: `relativepos="-10 10"` places stamp in lower right corner of page
`align center middle`	Align the stamp with the page center position horizontally at center of page, the x value of relativepos is ignored and should be set to 0. middle position vertically at middle of page, the y value of relativepos is ignored. Example 1: `position="0 4" align="center"` centers the stamp horizontally and 4pt away from the bottom of the page. Example 2: `position="-4 0" align="middle"` centers the stamp vertically and 4pt away from the right edge of the page.
`size w h`	The width and height of the stamp. The stamp's content will be clipped to this rectangle. If this is not specified or either w or h are zero, the respective size is calculated to fit content.

`rotate r`	Rotation of the stamp in degrees clockwise.
`scale relToA4`	Scale the stamp relative to the page size. i.e. make stamp half as large on a A5 and twice as large on a A3 page as specified.
`autoorientation true`	Detect orientation (portrait and landscape) of page automatically and treat landscape page as 90° rotated portrait. Useful to apply stamps to "long" or "short" edge of page.
`alpha t`	The opacity of the stamp as a whole. 1.0 for fully opaque, 0.0 for fully transparent. Default: 1.0 The PDF/A-1 standard does not allow transparency. Therefore, for PDF/A-1 conforming input files you must not set alpha to a value other than 1.0.
`type type`	The type of the stamp `annotation` (default): The stamp is added to the page as a stamp annotation. Creating or modifying stamps of this type will not invalidate existing signatures of the input document. While it is not easily possible to remove stamps of this type, it is possible to print a document without annotations. `foreground`[7]: The stamp is added to the foreground of the page content. Creating or modifying stamps of this type will invalidate all existing signatures of the input document. It is not easily possible to remove stamps of this type nor can the document be printed without them. `background`: The stamp is added to the background of the page content. Creating or modifying stamps of this type will invalidate all existing signatures of the input document. It is not easily possible to remove stamps of this type nor can the document be printed without them. Note that stamps placed this way can be hidden when pages contain a non-transparent background. In these cases, you may rather want to put the stamps in the foreground, but apply alpha transparency to achieve a result with existing content not covered completely.
`ps:rotate`	*Rotate*: Applies to stamp content defined within this tag
`angle n`	Rotate by n degrees counter-clockwise, e.g: 90
`origin x y`	Set the origin of the rotation in points, e.g. 100 100
`ps:translate`	*Coordinate Translation*: Applies to stamp content defined within this tag
`offset x y`	The x (horizontal) and y (vertical) offset in points. A translation by x y is equal to a transformation by 1 0 0 1 x y.
`ps:transform`	*Coordinate Transformation*: Applies to stamp content defined within this tag
`matrix a b c d x y`	The transformation matrix, to scale rotate, skew, translate, etc the stamp, e.g: Identity: 1 0 0 1 0 0 Scale by factor 2: 2 0 0 2 0 0 Translate 50 points left, 200 up: 1 0 0 1 50 200 Rotate by x: cos(x) sin(x) -sin(x) cos(x) 0 0 For 90 ($= \pi/2$) that is: 0 1 -1 0 0 0

[7] Up to version 4.5.6.0 of the 3-Heights™ PDF Security API this type was called content.

`ps:filltext`	*Add Filled Text*
`color r g b`	The color as RGB value, where all values must be from 0 to 1, e.g: Red: 1 0 0 Green: 0 1 0 Black: 0 0 0 Gray: 0.5 0.5 0.5
`position x y`	The position in points in the stamp, e.g. 200 300 With the default align values (`align="left top"`), position defines the left top corner of the text.[8]
`align xalign yalign`	Align text at position or stamp, if position is not set. Values for horizontal alignment xalign: - left align to the left (default) - center center text - right align to the right Values for vertical alignment yalign: - top align to the top (default) - middle align to the middle - bottom align to the bottom Example: `align="left bottom"` positions the text in the left bottom corner of the stamp. If position is set, align left bottom corner of text to position.
`font name`	The true type name of the font, e.g. `Arial` or `Times New Roman,Bold`, or a complete path to the font, e.g. `C:/Windows/Fonts/Arial.ttf`. If the name is used, the respective font must be available in any of the font directories (see chapter Fonts).
`size n`	The font size in points, e.g. 12. If set to 0 , the size is chosen such that text fits stamp size (not allowed, if operator is within transformation operator).
`fontencoding` `encoding`	Encoding of the font. Allowed values are Unicode (default) and WinAnsi, see note 3 below
`text text`	The text that is to be written, e.g. `text="Hello World"` Multiline text is supported by using the newline character `
`, e.g. `text="1st line
2nd line"`.
`ps:stroketext`	*Add Stroked Text* (Outlined Text): For parameters see filltext.
`linewidth f`	Set the linewidth in points, e.g. 1.0.
`color r g b`	See filltext
`position x y`	See filltext
`align xalign yalign`	See filltext
`font name`	See filltext
`size n`	See filltext
`fontencoding` `encoding`	See filltext
`text text`	See filltext

[8] Prior to version 4.4.31.0 of the 3-Heights™ PDF Security API, position specified the origin of the first character. When upgrading, add `0.75*size` to the value of y.

`ps:image`	*Add Image*: In order for the stamp to be PDF/A compliant, the image's color space must match the document's output intent's.
`rect x y w h`	The rectangle where the image is to be placed at. x,y correspond the the location (origin at lower left corner), and w,h to width and height, e.g: 100 200 50 50
`filename path`	The path to the file, e.g: C:\pictures\image1.jpg
`compression value`	By default bi-tonal images are compressed with CCITTFax, continuous tone images with DCT and indexed images with Flate. To explicitly set the compression use this property. Support values are: `Flate` Flate encoded `DCT` DCT (JPEG) encoded `CCITTFax` CCITT G4 encoded
`ps:fillrectangle`	*Add Filled Rectangle*:
`rect x y w h`	The coordinates and size of the rectangle. If this value is omitted, the rectangle fills the area of the stamp.
`color r g b`	The fill color of the rectangle. The color as RGB value, where all values must be from 0 to 1.
`alpha t`	The opacity of the rectangle. 1.0 for fully opaque, 0.0 for fully transparent. Default: 1.0 The PDF/A-1 standard does not allow transparency. Therefore, for PDF/A-1 conforming input files you must not set alpha to a value other than 1.0.
`ps:strokerectangle`	*Add Stroked Rectangle*:
`rect x y w h`	See fillrectangle
`linewidth f`	Set the linewidth in points, e.g. 1.0.
`color r g b`	The stroke color of the rectangle. The color as RGB value, where all values must be from 0 to 1.
`alpha t`	See fillrectangle

Notes

1. All coordinate and size values are in PDF units of 1/72 inch (A4 = 595 x 842 points, letter = 612 x 792 points). The origin of the coordinate system is generally the lower left corner of the reference object. For stamps the reference object is the page, for content operators the reference is the stamp rectangle.
2. Setting the name attribute of a stamp allows the stamp's content to be replaced later. If an existing stamp with the same name exists in the input file, its content is replaced. Otherwise a new stamp is created. Note that when updating a stamp, its position and size remains. Therefore, if you intend to update a stamp, make sure to create it using a size that is sufficiently large.
3. Attribute fontencoding: The PDF/A standard demands that all used fonts must be embedded in the PDF. Since fonts with many glyphs can be very large in size (>20MB), unused glyphs are removed prior to embedding. This process is called subsetting. The attribute fontencoding controls the subsetting:
   - Unicode: Only the glyphs used by the stamp are embedded. If the stamp is modified, a new font that includes the new glyph set has to be re-embedded. This setting is recommended for stamps that need not be modified later.
   - WinAnsi: All glyphs required for WinAnsiEncoding are embedded. Hence the text's characters must be limited to this character set. If the content of the stamp is updated, fonts using WinAnsi will be reused.

   For example, embedding the font "Arial" with Unicode and approximately ten glyphs uses 20KB while "Arial" with WinAnsi (approximately 200 glyphs) uses 53KB of font data.

Example: Simple Stamps

Apply two simple stamps.

First Stamp: Stamp text “Simple Stamp” on in upper right corner of all pages.

Second Stamp: Stamp image image.jpg rotated by 90° at the corner of the top edge of the first page.

```
<?xml version="1.0" encoding="utf-8"?>
<ps:pdfstamp xmlns:ps="http://www.pdf-tools.com/pdfstamp/">
  <ps:stamp page="all" name="simple stamp"
   relativepos="10 -10" size="160 0">
     <ps:filltext align="left middle" fontencoding="WinAnsi"
      font="Arial" size="12"
      text="Simple Stamp" />
  </ps:stamp>
  <ps:stamp page="first" relativepos="0 -10" align="center">
    <ps:rotate angle="90" origin="50 50">
      <ps:image rect="0 0 100 100"
       filename="c:\images\image.jpg"/>
    </ps:rotate>
  </ps:stamp>
</ps:pdfstamp>
```

Example: Modify Simple Stamp

Modify simple stamp from example above.

The stamp `simple stamp` can be modified by applying the following stamp xml file to the output file of the example above. Note that since position and size of the stamp remain unchanged, the respective attributes can be omitted.

```
<?xml version="1.0" encoding="utf-8"?>
<ps:pdfstamp xmlns:ps="http://www.pdf-tools.com/pdfstamp/">
  <ps:stamp name="simple stamp">
    <ps:filltext align="left middle" fontencoding="WinAnsi"
     font="Arial" size="12"
     text="Modified Stamp" />
  </ps:stamp>
</ps:pdfstamp>
```

Example: Add a watermark text diagonally across page

Note: an A4 page is 595 by 842 points.

```
<?xml version="1.0" encoding="utf-8"?>
<ps:pdfstamp xmlns:ps="http://www.pdf-tools.com/pdfstamp/">
  <ps:stamp page="all" size="595 842"
   align="center middle"
   scale="relToA4" autoorientation="true"
   type="foreground">
     <ps:rotate angle="55" origin="298 421">
       <ps:stroketext align="center middle" position="298 421"
        font="Arial,Bold" size="60"
        text="WATERMARK TEXT"/>
     </ps:rotate>
  </ps:stamp>
</ps:pdfstamp>
```

Example: Apply stamp to long edge of all pages

Stamp has a light gray background and a black border.

```
<?xml version="1.0" encoding="utf-8"?>
<ps:pdfstamp xmlns:ps="http://www.pdf-tools.com/pdfstamp/">
  <ps:stamp page="all" size="802 12"
   relativepos="5 0" align="middle" rotate="90"
   scale="relToA4" autoorientation="true"
   alpha="0.75" type="foreground">
     <ps:fillrectangle color="0.8 0.8 0.8" />
     <ps:strokerectangle />
     <ps:filltext align="center middle"
      font="Arial" size="8"
      text="stamp on long endge"/>
  </ps:stamp>
</ps:pdfstamp>
```

6 Reference Manual

Note this manual describes the COM interface only. Other interfaces (C, Java, .NET) however work similarly, i.e. they have calls with similar names and the call sequence to be used is the same as with COM.

6.1 PdfSecure Interface

AddDocMDP Signature

```
Method Boolean AddDocMDPSignature (PdfSignature pSignature, Short
    accessPermissions)
```

Add a document MDP (modification detection and prevention) signature. A PDF document can at most contain one MDP signature. A DocMDP signature defines the access permissions of the document. It should be combined with standard encryption, i.e. the function SaveAs should not apply encryption.

PDF documents with DocMDP signatures added with the 3-Heights™ PDF Security API require Acrobat 7 or later to be opened.

Parameters:

- pSignature: The digital signature that is to be added. The properties of the signature must be set before it is added.
- accessPermissions: The access permissions granted are one of the following three values:
  1. No changes to the document are permitted; any change to the document invalidates the signature.
  2. Permitted changes are filling in forms, instantiating page templates, and signing; other changes invalidate the signature.
  3. Permitted changes are the same as for 2, as well as annotation creation, deletion, and modification; other changes invalidate the signature.

Return value:

- True: Successfully added the signature to the document. Note: At this point it is not verified whether the certificate is valid or not. If an invalid certificate is provided the SaveAs function will fail later on.
- False: Otherwise

Note: This version can create visible DocMDP signatures. In order to create an invisible signature, set the signature's rectangle to [0, 0, 0, 0].

AddPreparedSignature

```
Method Boolean AddPreparedSignature(PdfSignature pSignature)
```

Add a signature field including an appearance but without a digital signature. This method must be called prior to SaveAs or SaveInMemory and should only be used in combination with SignPreparedSignature.

Parameters:
- pSignature:: The digital signature from which the field and appearance is created. The properties of the signature must be set before it is added.

Return value:
- True: Successfully prepared signature
- False: Otherwise

AddSignature

```
Method Boolean AddSignature(PdfSignature pSignature)
```

Add a digital signature to the document. The signature is defined using a PdfSignature object. This method must be called prior to SaveAs or SaveInMemory. Do not dispose of the PdfSignature object until the associated document has been saved or closed.

Parameters:
- pSignature: The digital signature that is to be added. The properties of the signature must be set before it is added.

Return value:
- True: Successfully added the signature to the document. Note: At this point it is not verified whether the certificate is valid or not. If an invalid certificate is provided the SaveAs function will fail later on.
- False: Otherwise

AddSignatureField

```
Method Boolean AddSignatureField(PdfSignature pSignature)
```

Add a signature field only. This method adds a field which is meant to be signed manual in a later step. This method must be called prior to SaveAs or SaveInMemory.

Parameters:
- pSignature: The digital signature that is to be added. The properties of the signature must be set before it is added.

Return value:
- True: Successfully added the signature to the document.
- False: Otherwise

AddStamps

```
Method Boolean AddStamps(String FileName)
```

Add a stamp xml file. This method must be called after the input file is opened and before the save operation. For more information about stamping, see the section Stamping.

AddStampsMem

```
Method Boolean AddStamps(Variant MemBlock)
```

Add a stamp xml from memory. This method must be called after the input file is opened and before the save operation. For more information about stamping, see the section Stamping.

AddTimeStampSignature

```
Method Boolean AddTimeStampSignature (PdfSignature pSignature)
```

Add a document level Time-stamp. No appearance is created. The following signature properties must be set: TimeStampURL. The following signature properties may be set: Provider, TimeStampCredentials, ProxyURL, ProxyCredentials.

BeginSession

```
Method Boolean BeginSession(String Provider)
```

The methods BeginSession and `EndSession` support bulk digital signing by keeping the session to the security device (HSM, Token or Cryptographic Provider) open. See the chapter Guidelines for Mass Signing for more guidelines.

For backwards compatibility the use of these methods is optional. If used, the Provider property may not be set. If omitted an individual session to the provider indicated by the property Provider is used for each signature operation.

Parameters:
- Provider: See property `Provider`.

Return value:
- True: Session started successfully
- False: Otherwise

Close

```
Method Boolean Close()
```

Close an opened input file. If the document is already closed the method does nothing.

Return value:
- True: The file was closed successfully.
- False: Otherwise

ErrorCode

```
Property TPDFErrorCode ErrorCode
Accessors: Get
```

This property can be accessed to receive the latest error code. See also enumeration TPDFErrorCode. PDF-Tools error codes are listed in the header file *pdferror.h*. Please note, that only few of them are relevant for the 3-Heights™ PDF Security API.

ErrorMessage

```
Property String ErrorMessage
Accessors: Get
```

Return the error message text associated with the last error (see property ErrorCode).

Note, that the property is NULL, if no message is available.

EndSession

```
Method Boolean EndSession()
```

Ends the opten session to the security device.

See BeginSession.

ForceEncryption

```
Property Boolean ForceEncryption
Accessors: Get, Set
Default: False
```

File encryption is not allowed by the PDF/A standard. Therefore 3-Heights™ PDF Security API aborts and returns an error, when encryption is configured and an input file is PDF/A. Use this option, in order to enable encryption of PDF/A conforming files. The conformance of the output file is downgraded to PDF.

ForceSignature

```
Property Boolean ForceSignature
Accessors: Get, Set
Default: False
```

Force signature allows DocMDP (PDF 1.6) and Time-stamp signatures (PDF 2.0) on PDF/A-1 documents. The output file's version is upgraded and PDF/A conformance removed. So the output file will contain the signature, but not be PDF/A-1 anymore.

Applying a DocMDP or Time-stamp signature breaks PDF/A-1 compliance, therefore the default behavior is to abort the operation with an error.

GetPDF

```
Method Variant GetPDF()
```

Get the output file from memory. See also method SaveInMemory.

Return value:
- A byte array containing the output PDF. In certain programming languages, such as Visual Basic 6, the type of the byte array must explicitly be Variant.

GetRevison

```
Method Variant GetRevision(Integer iRevision)
```

Return the PDF document of a given revision number. This is useful to retrieve the state of the PDF document at the time it has been signed. All revisions which had been applied after the given revision are ignored.

Return value:
- The selected revision of the PDF file

GetSignature

```
Method PdfSignature GetSignature(Long iSignature)
```

Get a signature field from the current document.

Parameters:
- iSignature: The selected signature in the document in the range from 0 to n-1, where 0 is the first and n-1 the last signature. The total number of signatures in the document can be retrieved using the property SignatureCount, which returns a value from 0 to n.

Return value:
- An interface to the PdfSignature

GetSignatureCount

Deprecated, use the property SignatureCount instead.

InfoEntry

```
Property String InfoEntry(String Key)
Accessors: Get, Set
```

Parameters:
- Key: A key as string

Return value:
- Value: The value as string

Retrieve or add a key-value pair to the documents info object. Values of predefined keys are also stored in the XMP metadata package.

Popular entries defined by the PDF Specification and used by most PDF viewers are “Title”, “Author”, “Subject”, “Creator” (sometimes referred to as Application) and “Producer” (sometimes referred to as PDF Creator).

Examples in Visual Basic 6:

- Get document title: `t = InfoEntry("Title")`
- Set document title: `InfoEntry( "Title ") = "my title"`
- Set the creation date to 13:55:33, April 5, 2010, UTC+2:
  `InfoEntry("CreationDate") = "D: 20100405135533 + 02'00'"`

NoCache

```
Property Boolean NoCache
Accessors: Get, Set
Default: False
```

Get or set whether to disable the cache for CRL and OCSP responses.

Using the cache is safe, since the responses are cached as long as they are valid only. The option affects both signature creation and validation.

See section on caching for more information on the caches.

Open

```
Method Boolean Open(String FileName, String Password)
```

Open a PDF file or raster image file, i.e. make the objects contained in the document accessible. If a document is already open, it is closed first.

Parameters:
- FileName: The file name and optionally the file path, drive or server string according to the operating systems file name specification rules.
- Password (optional): The user or the owner password of the encrypted PDF document. If this parameter is left out an empty string is used as a default.

Return value:
- True: The file could successfully be opened.
- False: The file does not exist, it is corrupt, or the password is not valid. Use the property ErrorCode for additional information.

OpenMem

```
Method Boolean OpenMem(Variant MemBlock, String Password)
```

Open a PDF document or raster image from memory, i.e. make the objects contained in the document accessible. If a document is already open, it is closed first.

Parameters:
- MemBlock: The memory block containing the PDF file given as a one dimensional byte array.
- Password (optional): The user or the owner password of the encrypted PDF document. If this parameter is left out an empty string is used as a default.

Return value:
- True: The document could successfully be opened.
- False: The document could not be opened, it is corrupt, or the password is not valid.

RevisionCount

```
Property Integer RevisionCount
Accessors: Get
```

Return the number of revisions of the document (the number of incremental updates). Although a linearized file looks like an incrementally updated file it only counts as one revision.

See also GetRevision.

SaveAs

```
Method Boolean SaveAs(String FileName, String UserPw, String OwnerPw,
    TPDFPermission PermissionFlags, Long KeyLength, String StrF, String StmF)
```

Create an output PDF document, apply the security settings and save the content from the input file to the output file.

The last three parameters (KeyLength, StrF, StmF) are only relevant in specific cryptographic situations. In all other cases, it is easiest to use the default values 128, “V2”, “V2”.

Parameters:
- FileName: The file name and optionally the file path, drive or server string according to the operating systems file name specification rules.
- UserPwd (optional): Set the user password of the PDF document. If this parameter is omitted, the default password is used. Use 0 to set no password.
- OwnerPwd (optional): Set the owner password of the PDF document. If this parameter is omitted, the default password is used. Use 0 to set no password.
- PermissionFlags (optional): The permission flags. By default no encryption is used (-1). The permissions that can be granted are listed at the enumeration TPDFPermission.
  To not encrypt the output document, set PermissionFlags to ePermNoEncryption (-1), user and owner password to 0.
  In order to allow high quality printing, flags ePermPrint and ePermDigitalPrint need to be set.

- KeyLength: (optional, Default: 128): The key length is a determining factor of the strength of the encrypting algorithm and the amount of time to break the cryptographic system. For RC4 the key length can be any value from 40 to 128 that is a multiple of 8. For AESV2 the key length is automatically set to 128, for AESV3 to 256. Notes:
  - Certain PDF viewers only support 40 and 128 bit encryption. Other tools, such as the 3-Heights™ tools also support other encryption key lengths
  - 256 bit encryption requires Acrobat 9 or later.
  - If the selected permission flags require a minimum key length, the key length is automatically adjusted (e.g. to 128 bits)
- StrF: (optional, Default: "V2"): Set the string crypt filter. Supported values are "None", "V2" (default), "RC4", "AESV2" and "AESV3". Setting this value to an empty string or null, means the default filter is used. Supported crypt filter:
  - "None": The application does not decrypt data
  - "V2" or "RC4": (PDF 1.2) The application asks the security handler for the encryption key and implicitly decrypts data using the RC4 algorithm.
  - "AESV2": (PDF 1.6) The application asks the security handler for the encryption key and implicitly decrypts data using the AES-V2 128 bit algorithm.
  - "AESV3": (PDF 1.7) The application asks the security handler for the encryption key and implicitly decrypts data using the AES-V3 256 bit algorithm.
- StmF: (optional, Default: "V2"): Set the stream crypt filter. Supported values are "None", "V2", "RC4", "AESV2" and "AESV3". Note that certain viewers require the stream crypt filter to be equal to the string crypt filter, e.g. both must be RC4 or AES. Other tools, such as the 3-Heights™ PDF tools do not have this limitation. Setting this value to an empty string or null, means the default filter is used.

Return value:
- True: The opened document could successfully be saved to file.
  Check the property ErrorCode after SaveAs to identify possible, non-critical issues during the process. These are[9]:
  - "SIG_CREA_E_OCSP": Couldn't get response from OCSP server.
  - "SIG_CREA_E_CRL": Couldn't get response from CRL server.
  - "SIG_CREA_E_TSP": Couldn't get response from Time-stamp server.

- False: Otherwise. One of the following occurred[10]:
  - The output file cannot be created
  - "SIG_CREA_E_SESSION": Cannot create a session (or CSP).
  - "SIG_CREA_E_STORE": The certificate store is not available.
  - "SIG_CREA_E_CERT": The certificate cannot be found.
  - "SIG_CREA_E_PRIVKEY": The private key is not available.
  - "SIG_CREA_E_INVCERT": The signing certificate is invalid, because it has expired, is not yet valid, or was revoked.
  - "PDF_E_SIGLENGTH": Incorrect signature length.

Set permission flags equally to Acrobat 7:

In Acrobat 7, there are four different fields/check boxes that can be set. In brackets are the corresponding permission flags.

[9] This is a complete list, all other values can be ignored.
[10] This is not a complete list. If SaveAs returs `False`, it is recommended to abort the processing of the file and log the error code and error message.

- Printing Allowed:
  - None ()
  - Low Resolution (ePermPrint)
  - High Resolution (ePermPrint + ePermDigitalPrint)

- Changes Allowed:
  - None ()
  - Inserting, deleting and rotating pages (ePermModify)
  - Filling in form fields and signing existing signature fields (ePermAnnotate)
  - Commenting, filling in form fields, and signing existing signature fields (ePermAnnotate + ePermFillForms)
  - Any except extracting pages (ePermModify + ePermAnnotate + ePermFillForms)
- Enable copying of text, images and other content (ePermCopy + ePermSupportDisabilities)
- Enable text access for screen reader devices for the visually impaired (ePermSupportDisabilities)

These flags can be combined. For example to grant permission which are equal to Acrobat's 7 "Printing Allowed: High Resolution" and "Enable copying of text, images and other content", set the flags ePermPrint + ePermCopy + ePermSupportDisabilities + ePermDigitalPrint.

SaveInMemory

```
Method Boolean SaveInMemory(String UserPw, String OwnerPw, TPDFPermission
    PermissionFlags, Long KeyLength, String StrF, String StmF)
```

Save the output PDF in memory. After the Close call it can be accessed using the method GetPDF.

All parameters are identical to the SaveAs method.

See also chapter How to use the in-Memory Functions.

Return value:
- True: The document could be saved in memory successfully.
- False: Otherwise

SetSessionProperty

```
Method Boolean SetSessionPropertyString(String Name, String Value)
Method Boolean SetSessionPropertyBytes(String Name, Variant Value)
```

Provider-specific session configuration.

Properties have to be set before calling BeginSession and are deleted when calling EndSession.

Parameters:
- Name: The name of the property. The names that are supported are specific to the provider used with BeginSession.
- Value: The value of the property as string or byte array.

SignatureCount

```
Property Long SignatureCount
Accessors: Get
```

Return the number of signature fields. If 0 is returned, it means there is no digital signature in the document.

SignPreparedSignature

```
Method Boolean SignPreparedSignature(PdfSignature pSignature)
```

Create a digital signature for an existing signature field, which was previously created using the method AddPreparedSignature. This method must be called prior to SaveAs or SaveInMemory.

Parameters:
- pSignature: The digital signature that is to be added. This must be the same signature as used in AddPreparedSignature.

Return value:
- True: Successfully added the signature to the document.
- False: Otherwise

SignSignatureField

```
Method Boolean SignSignatureField(PdfSignature pSignature)
```

Sign an empty signature field. An empty signature field can be added using AddSignatureField. This method must be called prior to SaveAs or SaveInMemory.

Parameters:
- pSignature: The digital signature that is to be added.

Return value:
- True: Successfully placed the signature into the signature field.
- False: Otherwise

Terminate

```
Method Void Terminate
```

Terminate all open sessions, and finalize and unload all PKCS#11 drivers. Some drivers require Terminate to be called. Otherwise, your application might crash and/or your HSM, USB token or smart card might not be unlocked.

When using the C/C++ API, Terminate may not be called from the context of the destructor of a global or static object, an atexit() handler, nor the DllMain() entry point.

Make sure to end all open sessions and dispose of all PdfSecure objects before calling Terminate. After calling Terminate, the process may not call any other methods of this class.

TestSession

```
Method Boolean TestSession()
```

Test if the current session is still alive.

Return value:
- True: Subsequent calls to SaveAs and SaveInMemory are likely to succeed.
- False: Subsequent calls to SaveAs and SaveInMemory are unlikely to succeed. Error codes are the same as in SaveAs where applicable.

ValidateSignature

```
Method Boolean ValidateSignature(PdfSignature pSignature)
```

Validate an existing digital signature, which was previously retrieved using the method `GetSignature`.

The component supports the verification of signatures including Time-stamps using cryptographic tokens and hardware security modules (HSM) through their PKCS#11 interface.

The validity checks are carried out at the time indicated either by the embedded time-stamp, if present, or by the signing time indicated in the PDF signature field object otherwise.

Furthermore, this method extracts the following values from the cryptographic signature and sets the respective properties of the `PdfSignature` object: `Date`, `Email`, `Name`, `Issuer`, `SignerFingerprint`, and `TimeStampFingerprint`.

If you get the error code `SIG_VAL_E_FAILURE`, your cryptographic provider does not offer the algorithms used for the signature. For example, the default provider (Microsoft CryptoAPI) does not support the SHA-2 hash algorithms. In this case, choose another provider.

Parameters:

- pSignature: The digital signature that is to be validated.

Return value:

- True: The digital signature is valid, i.e. the document has not been modified. If other problems are detected during signature validation, the property ErrorCode may have one of the following values:
  1. SIG_VAL_W_ISSUERCERT
  2. SIG_VAL_W_TSP
  3. SIG_VAL_W_TSPCERT
  4. SIG_VAL_W_NOTRUSTCHAIN
  5. SIG_VAL_W_PADES

  Note that the order of the list defines the priority of the error codes from highest to lowest. If multiple problems are detected, the error code with the highest priority is returned.

- False: The signature is corrupt or the document has been modified.

See also Enumeration TPDFErrorCode

6.2 PdfSignature Interface

This interface allows creating a signature and setting its position and appearance. The visual part of the signature consists of two (multi-line) texts. The string of both texts are generated automatically based on the signature properties if not set manually.

ContactInfo

```
Property String ContactInfo
Accessors: Get, Set
Default: ""
```

Add a descriptive text as signer contact info, e.g. a phone number. This enables a recipient to contact the signer to verify the signature. This is not required in order to create a valid signature.

If this property is set to an empty string, no entry is created.

Date

```
Property String Date
Accessors: Get, Set
Default: "D:00000000000000Z" (set to current date when signature is added)
```

This is the date when the signature is added. When this property is not set, the current time and date is used. The format of the date is: "D:YYYYMMDDHHMMSSZ". The meanings are:

D	Header of Date Format
YYYY	year
MM	month
DD	day
HH	hour
MM	minutes
SS	seconds
Z	UTC (Zulu) Time

Example for December 17, 2007, 14:15:13, GMT: "D:20071217141513Z".

Note: This property is set at the time when the signature is applied to the document.

If this property is set to an empty string, no entry is created.

DocumentHasBeenModified

```
Property Boolean DocumentHasBeenModified
Accessors: Get
```

Get whether the document has been modified (true) or not (false) since the selected signature was added.

Email

```
Property String Email
Accessors: Get
```

This property represents the email address of the signer. The method `ValidateSignature` extracts the address from the signing certificate's subject and sets this property.

EmbedRevocationInfo

```
Property Boolean EmbedRevocationInfo
Accessors: Get, Set
Default: True
```

Embed revocation information such as online certificate status response (OCSP - RFC 2560) and certificate revocation lists (CRL - RFC 3280).

Revocation information of a certificate is either an OCSP response or a CRL, which is provided by a validation service at the time of signing and acts as proof that at the time of signing the certificate is valid. This is useful because even when the certificates expires or is revoked at a later time, the signature in the signed document remains valid.

Embedding revocation information is optional but suggested when applying advanced or qualified electronic signatures. If the embedding is enabled then the information of the signer certificate and the issuer certificates

other than the root certificate is embedded as well. This implies that both OCSP responses and CRLs can be present in the same message.

The downsides of embedding revocation information are the increase of the file size (normally by around 20k) and that it requires a connection to a validation service, which delays the process of signing (normally by around 2 seconds). For mass signing it is suggested to use the caching mechanism, see chapter Caching of CRLs, OSCP and TSP Responses.

Embedding revocation information requires an online connection to the CA that issues them. The firewall must be configured accordingly. In case a web proxy is used, it must be ensured the following MIME types are supported when using OCSP (not required for CRL):

```
application/ocsp-request
```

```
application/ocsp-response
```

If EmbedRevocationInfo is set to true, but the embedding failed, e. g. because the OCSP server is not reachable, the return value of SaveAs is true, and the ErrorCode after SaveAs is SIG_CREA_E_OCSP.

FillColor

```
Property Long FillColor
Accessors: Get, Set
Default: 16761024 (red = 192, green = 192, blue = 255)
```

This property represents the color of the signature's background as an RGB value.

In order to not set a color, i.e. keep the rectangle transparent, set the FillColor to -1. This is particularly useful in combination with adding an image to the signature.

Filter

```
Property String Filter
Accessors: Get
```

Get the encryption filter of the signature, such as "Adobe.PPKLite".

FontName1

```
Property String FontName1
Accessors: Get, Set
Default: ""
```

This property represents the path to the font name used in upper text, i.e. the text that is set by the property Text1. Setting this property is optional on Windows and required on Non-Windows platforms.

Fontname2

```
Property String Fontname2
Accessors: Get, Set
Default: ""
```

This property represents the path to the font name used in lower text, i.e. the text that is set by the property Text2. Setting this property is optional on Windows and required on Non-Windows platforms.

FontSize1

```
Property Single FontSize1
Accessors: Get, Set
Default: 16
```

Define the font size of the Text1.

FontSize2

```
Property Single FontSize1
Accessors: Get, Set
Default: 8
```

Define the font size of the Text2.

HasSignature

```
Property Boolean HasSignature
Accessors: Get
```

Get whether the signature has an actual digital signature object or not.

ImageFileName

```
Property String ImageFileName
Accessors: Get, Set
Default: ""
```

Define the path to an image file that is to be added to the signature. The image is centered and scaled down proportionally to fit into the given rectangle. If the path is NULL, or the image does not exist, the appearance's background is a filled rectangle using the colors FillColor and StrokeColor. Note that for the output file to be PDF/A, the image's color space must match the document's output intent.

If you want the appearance to contain the image only and no text, set the property `Text2` to a space `" "`.

Issuer

```
Property String Issuer
Accessors: Get, Set
Default: ""
```

Set the issuer of the certificate. The "Issuer" corresponds to the common name (CN) of the issuer. In the Windows' certificate store this corresponds to "Issued by".

This property can be used to select the signer certificate for signing (see description of Cryptographic Provider in use).

LineWidth

```
Property Single LineWidth
Accessors: Get, Set
Default: 2
```

This is the thickness of the line surrounding the visual appearance of the signature.

Location

```
Property String Location
Accessors: Get, Set
Default: ""
```

This is the physical location where the signature was added, for example “Zurich, Switzerland”.

If this property is set to an empty string, no entry is created.

Name

```
Property String Name
Accessors: Get, Set
Default: ""
```

In order to sign a PDF document, a valid, existing certificate name must be provided.

The “Name” corresponds to the common name (CN) of the subject.

In the Windows’ certificate store this corresponds to “Issued to”.

When using a Windows OS, the certificate must be available in the Windows certificate store. See also chapter Digital Signature.

This property can be used to select the signer certificate for signing (see description of Cryptographic Provider in use).

PageNo

```
Property Long PageNo
Accessors: Get, Set
Default: -1 (last page)
```

Define the page number where the signature is to be added to the document. If an invalid page number is set, it is added to the last page.

Provider

```
Property String Provider
Accessors: Get, Set
Default (Windows only): "Microsoft Base Cryptographic Provider v1.0"
```

This property specifies the cryptographic provider used to create and verify signatures.

For more information on the different providers available, see the description in the respective subsection of the section Cryptographic Provider.

- When using the Microsoft CryptoAPI Provider, the value of this property with the following syntax:

  ```
  "[ProviderType:]Provider[;PIN]"
  ```

  Examples, “123456” being the pin code:
  - `Provider = "Microsoft Base Cryptographic Provider v1.0;123456"`
  - `Provider = ";123456"`

- When using the PKCS#11 Provider, the value of this property is to be set to a string with the following syntax:

  ```
  "PathToDll;SlotId;Pin"
  ```
  Examples:
  - `Provider = "\WINDOWS\system32\siecap11.dll;4;123456"`

- When using any of the service providers, such as the Swisscom All-in signing service, the value of this property is essentially the url of the service endpoint:

  ```
  "http[s]://server.servicedomain.com:8080/url"
  ```

ProxyURL

```
Property String ProxyURL
Accessors: Get, Set
Default: ""
```

This property has been deprecated. For more information, see the chapter How to Use a Proxy.

ProxyCredentials

```
Property String ProxyCredentials
Accessors: Get, Set
Default: ""
```

This property has been deprecated. For more information, see the chapter How to Use a Proxy.

Reason

```
Property String Reason
Accessors: Get, Set
Default: ""
```

Set or get the descriptive text for why the digital signature was added. It is not required in order to create a valid signature.

If this property is set to an empty string, no entry is created.

Rect

```
Property Variant Rect
Accessors: Get, Set
Default: [10, 10, 210, 60]
```

Set or get the position and size of the digital signature annotation. The default is in the lower left corner. The units are PDF points (1 point = 1/72 inch, A4 = 595x842 points, Letter = 612x792 points) measured from the lower left corner of the page.

The position is defined by the four values for the lower-left (x1, y1) and upper-right (x2, y2) corner of the rectangle. If either the width or height is zero or negative, an invisible signature is created, i.e. no visible appearance is created for the signature. To create an invisible signature set the rectangle to `[0, 0, 0, 0]`.

Hint about using this property in programming language that do not support the type "Variant": In order to find out what type you should use, create a PdfSignature object and look at the default value of the property in the debugger.

Revision

```
Property Integer Revision
Accessors: Get
```

Return the revision number of the PDF document associated with this signature. The associated PDF document can be retrieved using the method GetRevision.

SerialNumber

```
Property String SerialNumber
Accessors: Get, Set
```

The serial number with the issuer can be used to select a certificate for signing. This property is a hex string as displayed by the "Serial number" field in the Microsoft Management Console (MMC), e.g. "49 cf 7d d1 6c a9".

This property can be used to select the signer certificate for signing (see description of Cryptographic Provider in use).

SignerFingerprint

```
Property Variant SignerFingerprint
Accessors: Get, Set
```

The sha1 fingerprint of the signer certificate. This property can be used to select the signer certificate for signing (see description of Cryptographic Provider in use). After validating a signature, this property contains the validated signature's fingerprint.

SignerFingerprintStr

```
Property String SignerFingerprintStr
Accessors: Get, Set
```

The hex string representation of the signer certificate's sha1 fingerprint. This property can be used to select the signer certificate for signing (see description of Cryptographic Provider in use).

All characters outside the ranges 0-9, a-f and A-F are ignored. In the Microsoft Management Console, the "Thumbprint" value can be used without conversion, if the "Thumbprint algorithm" is "sha1". E.g. "b5 e4 5c 98 5a 7e 05 ff f4 c6 a3 45 13 48 0b c6 9d e4 5d f5".

Store

```
Property String Store
Accessors: Get, Set
Default: "MY"
```

For the Microsoft CryptoAPI Provider this defines the certificate store from where the signing certificate should be taken. This depends on the OS. The default is "MY". Other supported values are: "CA" or "ROOT".

StoreLocation

```
Property Integer StoreLocation
Accessors: Get, Set
Default: 1
```

For the Microsoft CryptoAPI Provider this defines the location of the certificate store from where the signing certificate should be taken. Supported are:

0 Local Machine

1 Current User (default)

For more information, see the detailed description of the Microsoft CryptoAPI Provider.

StrokeColor

```
Property Long StrokeColor
Accessors: Get, Set
Default: 8405056 (red = 64, green = 64, blue = 128)
```

This is the color of the signature's border line as an RGB value.

In order to not set a color, i.e. keep it transparent, set the StrokeColor to -1.

SubFilter

```
Property String SubFilter
Accessors: Get
```

Get the name of the sub filter, such as "adbe.pkcs7.detached".

Text1

```
Property String Text1
Accessors: Get, Set
Default: ""
```

This is the upper text that is added to the signature.

If this property is set to blank, the signature name is added to the upper text line of the visual signature.

In order to position text use the following syntax: <tab><x>,<y><delimiter><text>

`tab`	tabulator
`x, y`	integers
`delimiter`	Single character such as space
`text`	Any text string not containing a <tab>

Example for Visual Basic .NET:

```
Dim sig As New PdfSecure.Signature
...
sig.Text1 = Microsoft.VisualBasic.vbTab & "5,50 Peter Pan"
sig.Text2 = Microsoft.VisualBasic.vbTab & "15,25 Signed this document"
```

Text2

```
Property String Text2
Accessors: Get, Set
Default: ""
```

This is the lower text that is added to the signature. The text can be multi-lined by using linefeed ('\n', 0xA).

If this property is set to blank, a text three-line text is constructed that consists of:

- A statement who applied to signature
- The reason of the signature
- The date

See also property Text1.

If you want the appearance to not contain any text, set this property to a space `" "`.

TimeStampCredentials

```
Property String TimeStampCredentials
Accessors: Get, Set
Default: ""
```

If a Time-stamp server requires authentication, use this property to provide the credentials. Credentials commonly have the syntax “username:password”.

TimeStampFingerprint

```
Property Variant TimeStampFingerprint
Accessors: Get
```

The sha1 fingerprint of the Time-stamp server certificate. After validating a signature that contains a Time-stamp, this property contains the fingerprint of the Time-stamp server’s certificate.

TimeStampURL

```
Property String TimeStampURL
Accessors: Get, Set
Default: ""
```

The URL of the trusted Time-stamp authority (TSA) from which a Time-stamp shall be acquired. This setting is suggested to be used when applying a Qualified Electronic Signature. Example: “tsu.my-timeserver.org”. Applying a Time-stamp requires an online connection to a time server; the firewall must be configured accordingly. In case a web proxy is used, it must be ensured the following MIME types are supported:

`application/timestamp-query`

`application/timestamp-reply`

If an invalid Time-stamp server address is provided or no connection can be made to the time server, the return code of SaveAs is true, and the property ErrorCode is set to SIG_CREA_E_TSP after calling SaveAs.

6.3 Enumerations

Note: Depending on the interface, enumerations may have “TPDF” as prefix (COM, C) or “PDF” as prefix (.NET) or no prefix at all (Java).

TPDFErrorCode

All TPDFErrorCode enumerations start with “PDF_” followed by a single letter which is one of “S”, “E”, “W” or “I”, an underscore and a descriptive text. The single letter gives in an indication of the type of error. These are: Success, Error, Warning, Information. With respect to corrupt PDF files: An error indicates a corruption in the PDF, the file may or may not be readable. A warning indicates the file is readable but not valid.

A full list of all PDF Tools error codes is available in the header file *pdferror.h*. Note that only a few are relevant for the PDF Security API. The most common are listed here.

`PDF_S_SUCCESS`	The operation was completed successfully.
`LIC_E_NOTSET`, `LIC_E_NOTFOUND`, …	Various license management related errors.
`PDF_E_FILEOPEN`	Failed to open the file.
`PDF_E_FILECREATE`	Failed to create the file.
`PDF_E_PASSWORD`	The authentication failed due to a wrong password.
`PDF_E_UNKSECHANDLER`	The file uses a proprietary security handler, e.g. for a proprietary digital rights management (DRM) system.
`PDF_E_XFANEEDSRENDERING`	The file contains unrendered XFA form fields, i.e. the file is an XFA and not a PDF file.
`PDF_W_ENCRYPT`	Aborted processing of signed and encrypted document.
`SIG_CREA_E_SESSION`	Cannot create a session (or CSP).
`SIG_CREA_E_STORE`	Cannot open certificate store.
`SIG_CREA_E_CERT`	Certificate not found in store.
`SIG_CREA_E_INVCERT`	The signing certificate is invalid.
`SIG_CREA_E_OCSP`	Couldn’t get response from OCSP server.
`SIG_CREA_E_CRL`	Couldn’t get response from CRL server.
`SIG_CREA_E_TSP`	Couldn’t get response from Time-stamp server.
`SIG_CREA_E_PRIVKEY`	Private key not available. This is usually because a pin is required and was not entered correctly.
`SIG_CREA_E_SERVER`	Server error:
`SIG_CREA_E_ALGO`	The cryptographic provider does not implement a require algorithm.
`SIG_CREA_E_FAILURE`	Another failure occurred.
`PDF_E_SIGLENGTH`	Incorrect signature length.

Validation specific error codes:

SIG_VAL_E_ALGO	Unsupported algorithm found.
SIG_VAL_E_FAILURE	Program failure occurred.
SIG_VAL_E_CMS	Malformed cryptographic message syntax (CMS).
SIG_VAL_E_DIGEST	Digest mismatch (document has been modified).
SIG_VAL_E_SIGNER CERT	Signer's certificate is missing.
SIG_VAL_E_SIGNATURE	Signature is not valid.
SIG_VAL_W_ISSUER CERT	None of the certificates was found in the store.
SIG_VAL_W_NOTRUST CHAIN	The trust chain is not embedded.
SIG_VAL_W_TSP	The Time-stamp is invalid.
SIG_VAL_W_TSPCERT	The Time-stamp certificate was not found in the store.
SIG_VAL_W_NOTSP	The Time-stamp is not present.
SIG_VAL_W_PADES	The signature does not conform to the PAdES standard, e.g. because the signature is not DER encoded or the CMS contains more than one SignerInfo[11].

TPDFPermission

An enumeration for permission flags. If a flag is set, the permission is granted.

ePermNoEncryption	Do not apply encryption. This enumeration shall not be combined with another enumeration. When using this enumeration set both passwords to an empty string or null.
ePermAll	Grant all Permissions
ePermPrint	Low resolution printing
ePermModify	Changing the document
ePermCopy	Content copying or extraction
ePermAnnotate	Annotations
ePermFillForms	Filling of form fields
ePermSupportDisabilities	Support for disabilities
ePermAssemble	Document Assembly
ePermDigitalPrint	High resolution printing

Changing permissions or granting multiple permissions is done using a bitwise or operator. Changing the current permissions in Visual Basic should be done like this:

Allow Printing:	`Permission = Permission Or ePermPrint`
Prohibit Printing:	`Permission = Permission And Not ePermPrint`

[11] Adobe Acrobat XI classifies such signatures as valid.

7 Licensing and Copyright

The 3-Heights™ PDF Security API is copyrighted. This user's manual is also copyright protected; it may be copied and given away provided that it remains unchanged including the copyright notice.

8 Contact

PDF Tools AG

Kasernenstrasse 1

8184 Bachenbülach

Switzerland

http://www.pdf-tools.com